File này sẽ đề cập đến các vấn đề sau:

- **Định nghĩa của path, guide.**
- **Phân tích (nếu có), đưa ra ví dụ minh họa cho tất cả các lệnh liên quan.**
- **Phân biệt path và guide**, cụ thể là

In Asymptote, a path is a evaluated cubic spline, but a guide is not evaluated. You set every node of the spline, and the tangent direction, but you didn't specify the control points between nodes. Note that four points (two nodes and two control points) determine a Bezier segment. As a suggestion, you should always use guide to build a curve in a loop. (Leo Liu)

Tài liệu tham khảo:
A User's Manual for MetaPost
path.h
runpath.in
plain_paths.asy

TODO:
Trong runpath.in, có
path g; thì g được khởi tạo ngầm là nullpath
bool ==(path a, path b); kiểm tra 2 path có đồng nhất không.
bool !=(path a, path b); giá trị logic ngược lại với bool ==(path a, path b);

Trong plain_paths.asy, còn có vài lệnh mà tài liệu chính chưa đề cập, nên bổ sung nó.
Gồm có:
arcdir
reldir

Tại sao các lệnh vẽ đồ thị lại dùng kiểu dữ liệu là guide mà không phải path?

Số trang dự tính 80

Path là "nét vẽ" trong Asymptote, nó có thể là **điểm** (vì pair có thể **cast** được thành path), là **đoạn thẳng**, là **đa giác**, hoặc là **một đường cong bất kỳ**.

Path: a cubic spline resolved into a fixed path. The implicit initializer for paths is ***nullpath***.
Giá trị khởi tạo ngầm cho các path là nullpath.

Ví dụ, lệnh vẽ **đường tròn** tùy ý dựa trên đường tròn đơn vị (**unitcircle**)
path **circle(pair c, real r)** // pair c : tâm đường tròn, real r : bán kính
{
 return shift(c)*scale(r)*unitcircle;
}
If high accuracy is needed, a true circle may be produced with the routine **Circle** defined in the module graph:
import graph;
*path **Circle(pair c, real r, int n=nCircle)**;*

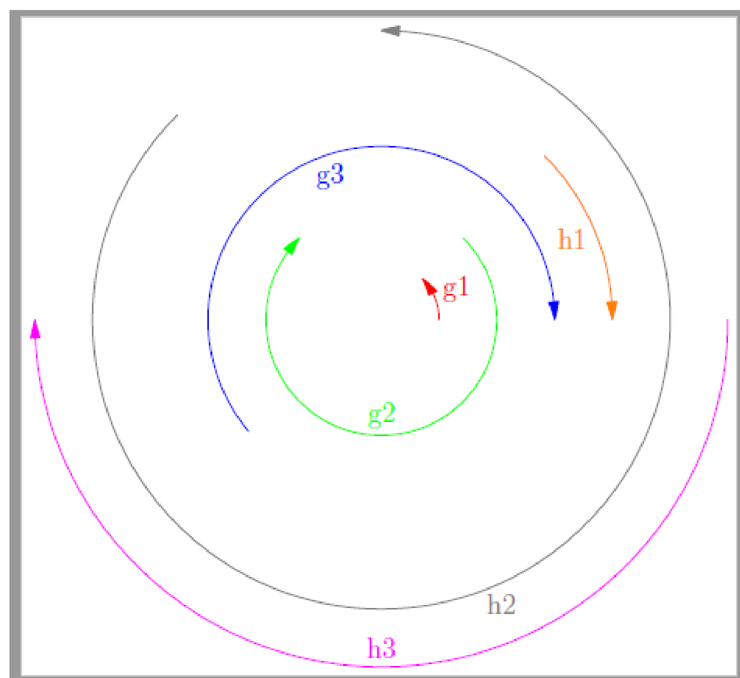
Cung tròn (plain_arcs.asy)
Trong Asymptote, có các hàm để vẽ cung tròn như sau:

- path **arc(pair c, real r, real angle1, real angle2)**;

Lệnh này vẽ một cung tròn tâm c, bán kính r từ góc angle1 đến angle2; vẽ theo chiều ngược chiều kim đồng hồ nếu angle2 >= angle1.

- path **arc(pair c, real r, real angle1, real angle2, bool direction)**;

// Lệnh này có thêm tùy chọn cho hướng của path.
//CCW (counter-clockwise) – ngược chiều kim, CW (clockwise) – cùng chiều kim.
Ví dụ 1:
size(300);
pair A=(0,0);
real



a=0.5,b=1,c=1.5,d=2,e=2.5,f=3;
path g1=arc(A,a,0,45,CCW);
path g2=arc(A,b,45,135,CW);

path g3=arc(A,c,220,360,CW);
path h1=arc(A,d,45,0);
path h2=arc(A,e,135,90,CCW);
path h3=arc(A,f,360,180);

draw(g1,red,Arrow);
draw(g2,green,Arrow);
draw(g3,blue,Arrow);
draw(h1,orange,Arrow);
draw(h2,gray,Arrow);
draw(h3,magenta,Arrow);
shipout(bbox(1mm,invisible));

- path **arc(pair c, explicit pair z1, explicit pair z2, bool direction=CCW)**;

// Chú ý là giả định |z1-c|=|z2-c|(cùng bán kính)
// Vẽ một cung có tâm ở c từ điểm z1 đến điểm z2 ngược chiều kim đồng hồ.
Nếu cần độ chính xác cao, trong module graph, chúng ta có các hàm vẽ cung sau:
import graph;
path Arc(pair c, real r, real angle1, real angle2, bool direction, int n=nCircle);
path Arc(pair c, real r, real angle1, real angle2, int n=nCircle);
path Arc(pair c, explicit pair z1, explicit pair z2, bool direction=CCW, int n=nCircle);

Một **elip** có thể được vẽ với hàm sau:

- path **ellipse(pair c, real a, real b)**{ return shift(c)*scale(a,b)*unitcircle; }

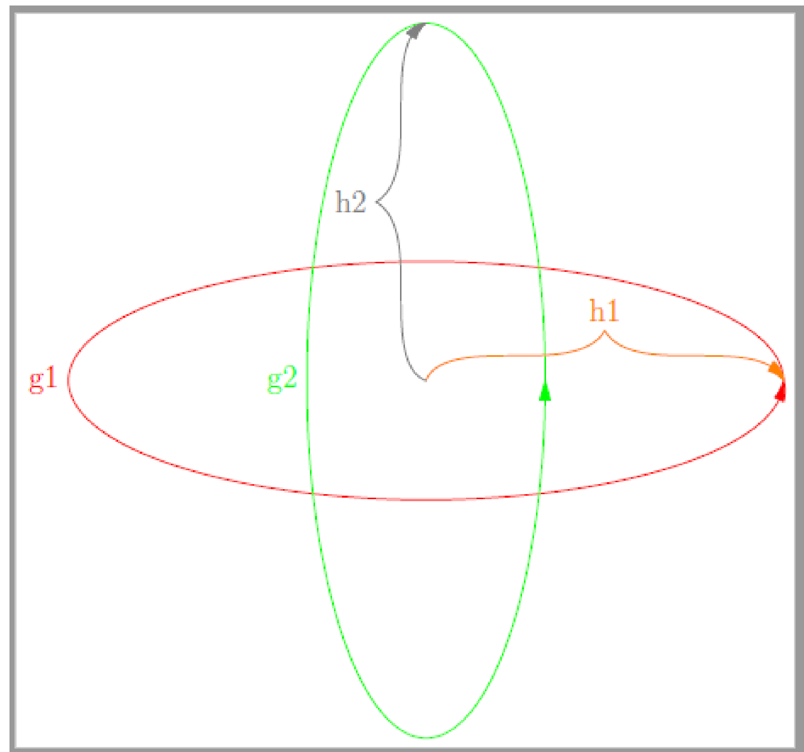
trong đó c là tâm ellipse, nếu a>b thì a là trục lớn, ngược lại a là trục bé; nếu a=b, ellipse trở thành đường tròn.

Một **brace(dấu ngoặc)** có thể được vẽ giữa hai tọa độ với lệnh:

- path **brace(pair a, pair b, real amplitude=bracedefaultratio*length(b-a))**;

Lệnh này nằm trong plain_paths.asy.
Ví dụ 2:

```
size(300);
pair A=(0,0);
real a=3,b=1;
path g1=ellipse(A,a,b);
path g2=ellipse(A,b,a);
path h1=brace(A,(a,0));
path h2=brace(A,(0,a));
draw(g1,red,Arrow);
draw(g2,green,Arrow);
draw(h1,orange,Arrow);
draw(h2,gray,Arrow);
```




```
shipout(bbox(1mm,invisible));
```

Here are some useful functions for paths: (Ở đây là một vài hàm hữu ích cho path)

- int **length(path p)**;

This is the number of (linear or cubic) segments in path p. If p is cyclic, this is the same as the number of nodes in p.
Đây là số đoạn thẳng trong path p. Nếu p là cyclic (kín), đây thì giống như số nodes trong p.

- int **size(path p)**;

This is the number of nodes in the path p. If p is cyclic, this is the same as length(p).
Đây là số nodes trong path p. Nêu p là cyclic, đây thì giống như **length(p)**.

Ví dụ 3:

path g=(0,0)--(4,0)--(4,3);
path k=g--cycle; // k is cyclic
path d=(0,0)..controls (0,1) and (1,1)..(1,0); // d is a Bezier curve
path h=(0,0)..(1,0)..(1,1)..(0,1); // d is cubic spline

write(length(g)); // 2
write(length(k)); // 3
write(length(d)); // 1
write(length(h)); // 3
write(size(g)); // 3
write(size(k)); // 3
write(size(d)); // 2 (ở đây chỉ có 2 node là (0,0) và (1,0), xem 5 Bezier curves để hiểu)
write(size(h)); // 4

- bool **cyclic(path p)**;

returns true iff path p is cyclic.

- bool **straight(path p, int i)**;

returns true iff the segment of path p between node i and node i+1 is straight.

- bool **piecewisestraight(path p)**;

returns true iff the path p is piecewise straight.

Ví dụ 4:

path g=(0,0)--(4,0)--(4,3);
path k=g--cycle; // k is cyclic
path d=(0,0)..controls (0,1) and (1,1)..(1,0); // d is a Bezier curve
path h=(0,0)..(1,0)..(1,1)..(0,1); // d is cubic spline

write(cyclic(g)); // false
write(cyclic(k)); // true
write(cyclic(d)); // false
write(cyclic(h)); // false
write(piecewisestraight(g)); // true
write(piecewisestraight(k)); // true
write(piecewisestraight(d)); // false
write(piecewisestraight(h)); // false

write(straight(g,1)); // true
write(straight(k,2)); // true
write(straight(d,1)); // false

- pair **point(path p, int t)**;

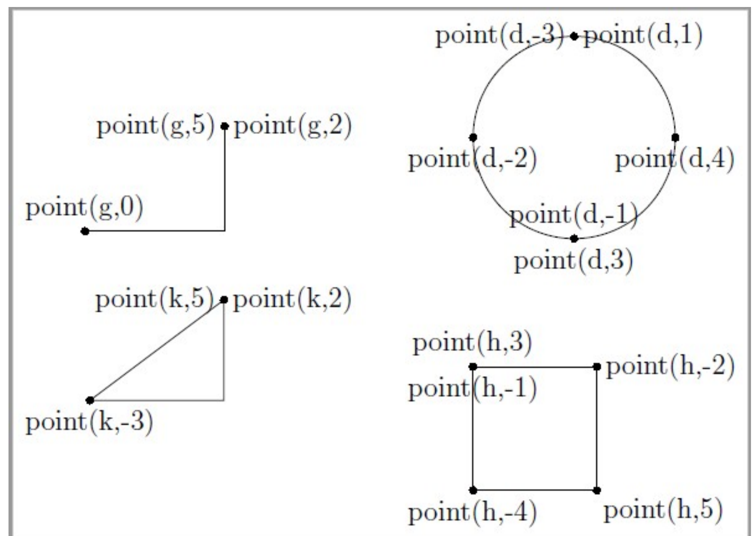
If p is cyclic, return the coordinates of node t mod length(p). Otherwise, return the coordinates of node t, unless t < 0 (in which case point(0) is returned) or t > length(p) (in which case point(length(p)) is returned).
Nếu p là kín, trả lại tọa độ của node t đồng dư length(p). Điều này có nghĩa là **point(p,t1) ≡ point(p,t2) (mod length(p))**. Ngược lại (*nếu p không cyclic*), trả lại tọa độ của node t, trừ khi t<0 (trong trường hợp này point(0) được trả lại) hay t> length(p) (trong trường hợp này point(length(p)) được trả lại).

- pair **point(path p, real t)**;

This returns the coordinates of the point between node floor(t) and floor(t)+1 corresponding to the cubic spline parameter t-floor(t) (see Chapter 5 [Bezier curves], page 22). If t lies outside the range [0,length(p)], it is first reduced modulo length(p) in the case where p is cyclic or else converted to the corresponding endpoint of p.
Lệnh này trả lại tọa độ của điểm giữa node floor(t) and floor(t)+1 tương ứng với tham số cubic spline t-floor(t). Nếu t nằm ngoài phạm vi [0,length(p)], nó đầu tiên bị quy về modulo length(p) trong trường hợp p là cyclic hay ngược lại (*nếu p không cyclic*) bị chuyển đổi thành tương ứng với điểm cuối của p (một path không cyclic thì luôn có 2 điểm endpoint).

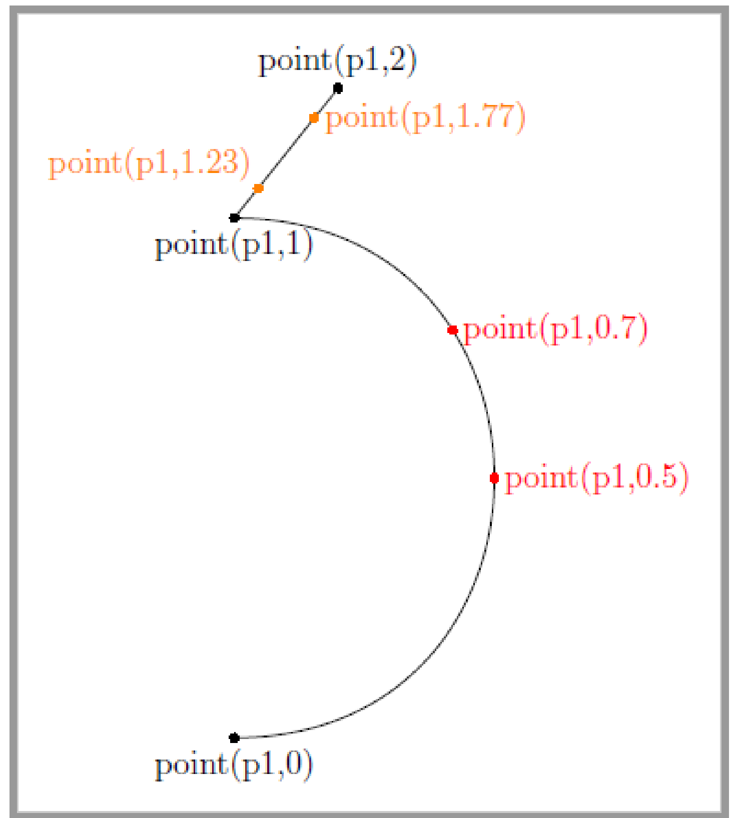
Ví dụ 5:
a.

```
path g=(0,0)--(4,0)--(4,3);
path k=g--cycle; // k is cyclic
path d=unitcircle;
path h=unitsquare;
picture pic1,pic2,pic3,pic4;
size(pic1,5cm);
draw(pic1,g);
dot(pic1,"point(g,2)",point(g,2));
```




```
dot(pic1,"point(g,0)",point(g,0),dir(90));
dot(pic1,"point(g,5)",point(g,5),dir(180));
size(pic2,5cm);
draw(pic2,k);
dot(pic2,"point(k,2)",point(k,2));
dot(pic2,"point(k,-3)",point(k,-3),dir(-90));
dot(pic2,"point(k,5)",point(k,5),dir(180));
```

```
size(pic3,5cm);
draw(pic3,d);
dot(pic3,"point(d,-3)",point(d,-3),dir(180));
dot(pic3,"point(d,-2)",point(d,-2),dir(-90));
dot(pic3,"point(d,-1)",point(d,-
1),dir(90));
dot(pic3,"point(d,1)",point(d,1));
```




```
dot(pic3,"point(d,3)",point(d,3),dir(-90));
dot(pic3,"point(d,4)",point(d,4),dir(-90));
size(pic4,5cm);
draw(pic4,h);
dot(pic4,"point(h,-2)",point(h,-2));
dot(pic4,"point(h,-1)",point(h,-1),dir(-90));
dot(pic4,"point(h,3)",point(h,3),dir(90));
dot(pic4,"point(h,-4)",point(h,-4),dir(-90));
dot(pic4,"point(h,5)",point(h,5),dir(-45));
picture pic;
add(pic,pic1.fit(),(0,0),10N);
add(pic,pic2.fit(),(0,0),10S);
picture picc;
add(picc,pic3.fit(),(0,0),10N);
add(picc,pic4.fit(),(0,0),10S);
add(pic.fit(),(0,0),10W);
add(picc.fit(),(0,0),10E);
shipout(bbox(1mm,invisible));
```

b.

```
size(10cm);
```

```
path p1 = (0,-1){E}..{W}(0,1)--(0.4,1.5);
draw(p1);
dot(Label("point(p1,0)"), point(p1,0), dir(-90), black);
dot(Label("point(p1,0.5)"), point(p1,0.5), E, red);
dot(Label("point(p1,0.7)"), point(p1, 0.7), red);
dot(Label("point(p1,1)"), point(p1,1), dir(-90), black);
dot(Label("point(p1,1.23)"), point(p1,1.23), dir(135), orange);
dot(Label("point(p1,1.77)"), point(p1,1.77), E, orange);
dot(Label("point(p1,2)"), point(p1,2), dir(90), black);
shipout(bbox(3mm,invisible));
```

- pair **dir(path p, int t, int sign=0, bool normalize=true)**;

If sign < 0, return the direction (as a pair) of the incoming tangent to path p at node t; if sign > 0, return the direction of the outgoing tangent. If sign=0, the mean of these two directions is returned.

- pair **dir(path p, real t, bool normalize=true)**;

returns the direction of the tangent to path p at the point between node floor(t) and floor(t)+1 corresponding to the cubic spline parameter t-floor(t) (see Chapter 5 [Bezier curves], page 22).

- pair **dir(path p)**;

returns dir(p,length(p)).

- pair **dir(path p, path q)**;

returns unit(dir(p)+dir(q)).

Ví dụ 6:

- pair **accel(path p, int t, int sign=0)**;

If sign < 0, return the acceleration of the incoming path p at node t; if sign > 0, return the acceleration of the outgoing path. If sign=0, the mean of these two accelerations is returned.

- pair **accel(path p, real t)**;

returns the acceleration of the path p at the point t.

- real **radius(path p, real t)**;

returns the radius of curvature of the path p at the point t.

Ví dụ 7:

- pair **precontrol(path p, int t)**;

returns the precontrol point of p at node t.
trả lại điểm precontrol của p tại node t.

- pair **precontrol(path p, real t)**;

returns the effective precontrol point of p at parameter t.
trả lại điểm precontrol hữu hiệu của p tại tham số t.

- pair **postcontrol(path p, int t)**;

returns the postcontrol point of p at node t.
trả lại điểm postcontrol của p tại node t.

- pair **postcontrol(path p, real t)**;

returns the effective postcontrol point of p at parameter t.
trả lại điểm postcontrol hữu hiệu của p tại node t.

Ví dụ 8:
Finding the Control Points of a Bezier Curve
https://javascript.info/bezier-curve
https://tex.stackexchange.com/a/178978/219419

- real **arclength(path p)**;

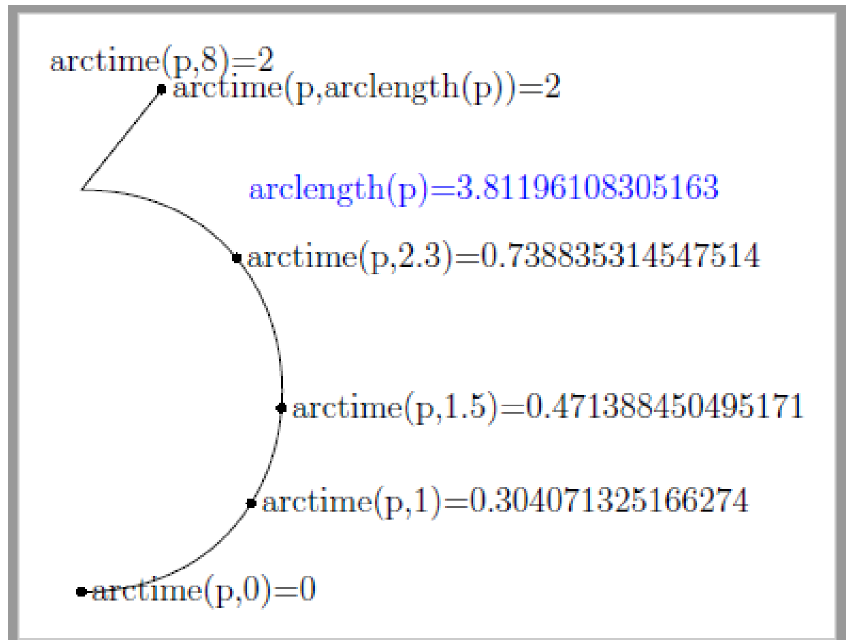
returns the length (in user coordinates) of the piecewise linear or cubic curve that path p represents.
trả lại độ dài (theo tọa độ người dùng) của đoạn thẳng hay đường cong cubic mà p biểu diễn.

- real **arctime(path p, real L)**;

returns the path "time", a real number between 0 and the length of the path in the sense of point(path p, real t), at which the cumulative **arclength** (measured from the beginning of the path) equals L.
trả lại "thời điểm" path, một số thực giữa 0 và chiều dài của path theo nghĩa của point(path p, real t), ở tại đó **chiều dài cung** tích lũy (được đo từ phần bắt đầu của path) bằng L.

- pair **arcpoint(path p, real L)**;

returns point(p,arctime(p,L)).
Ví dụ 9:
size(10cm);
path p = (0,-1){E}..{W}(0,1)--(0.4,1.5);
draw(p);
dot("arctime(p,0)"+"$=$"+(string) arctime(p,0),arcpoint(p,0));
dot("arctime(p,1)"+"$=$"+(string) arctime(p,1),arcpoint(p,1));



dot("arctime(p,1.5)"+"$=$"+(string) arctime(p,1.5),arcpoint(p,1.5));
dot("arctime(p,2.3)"+"$=$"+(string) arctime(p,2.3),arcpoint(p,2.3));
dot("arctime(p,arclength(p))"+"$=$"+(string) arctime(p,arclength(p)),arcpoint(p,arclength(p)));
dot("arctime(p,8)"+"$=$"+(string) arctime(p,8),arcpoint(p,8),dir(90));
label("arclength(p)"+"$=$"+(string) arclength(p),(2,1),blue);
shipout(bbox(3mm,invisible));
Trả lại các "thời điểm" path giữa 0 và 2 (2 được hiểu là chiều dài của path theo nghĩa point(path p, real t)), còn L nằm trong phạm vi tối thiểu từ 0 đến arclength(path p), chữ tích lũy có nghĩa là nếu L vượt ngoài arclength(path p), như trong ví dụ là arctime(p,8), thì nó vẫn trùng với arctime(p, arclength(path p)).
Để hiểu rõ ràng hơn lệnh **arctime(path p, real L)**, xem ví dụ 11.

- real **dirtime(path p, pair z)**;

returns the first "time", a real number between 0 and the length of the path in the sense of point(path, real), **at which the tangent to the path has the direction of pair z, or -1 if this never happens**.

trả lại “thời điểm” path đầu tiên, một số thực giữa 0 và chiều dài của path theo nghĩa point(path p, real t), ở tại đó tiếp tuyến đến path có hướng của tọa độ z, hoặc -1 nếu điều này không xảy ra.

Ví dụ 10:

// fig_pp01_290410_points_particuliers.asy

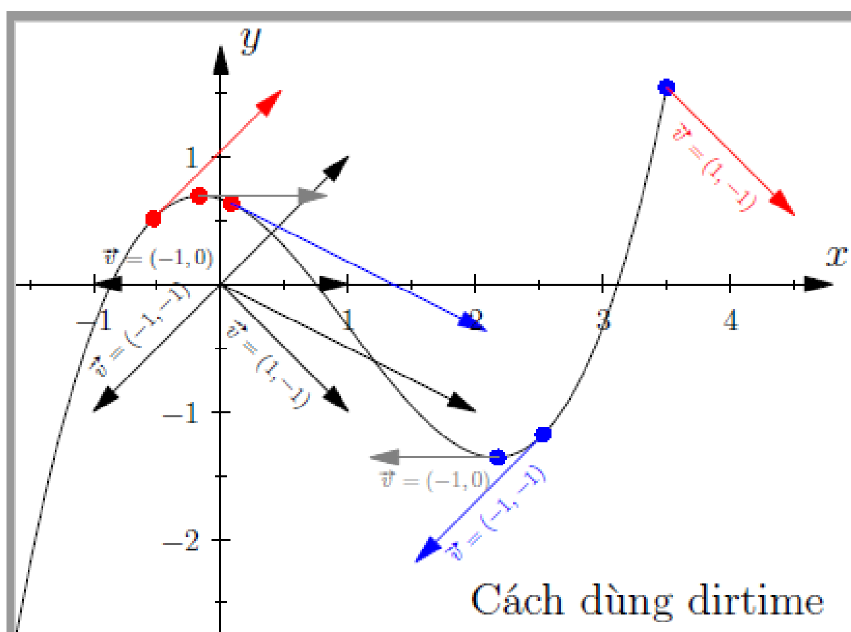
```
import graph;
usepackage("vietnam","utf8");
usepackage("esvect");
size (8cm);

real f(real x) {return (x^3-3x^2-x+2)/3;}
path Cf=graph(f,-1.6,3.5);
draw(Cf);

real tA=dirtime(Cf,(1,0));
real tB=dirtime(reverse(Cf),(-1,0));
real mA=dirtime(Cf,(1,1));
real mB=dirtime(reverse(Cf),(1,-1));
real nA=dirtime(Cf,(2,-1));
real nB=dirtime(reverse(Cf),(-1,-1));
draw((0,0)--(1,0),Arrow);
draw(rotate(180+degrees((-1,0)))*scale(.5)*Label("$\vv{v}=(-1,0)$"),(0,0)--
(-1,0),Arrow);
draw((0,0)--(1,1),Arrow);
draw(rotate(degrees((1,-1)))*scale(.5)*"$\vv{v}=(1,-1)$",(0,0)--(1,-1),Arrow);
draw((0,0)--(2,-1),Arrow);
draw(rotate(180+degrees((-1,-1)))*scale(.5)*"$\vv{v}=(-1,-1)$",(0,0)--(-1,-
1),Arrow);
pair pA=point(Cf,tA);
pair pB=point(reverse(Cf),tB);
pair mpA=point(Cf,mA);
pair mpB=point(reverse(Cf),mB);
pair npA=point(Cf,nA);
pair npB=point(reverse(Cf),nB);

dot(pA,4bp+red);
dot(pB,4bp+blue);
dot(mpA,4bp+red);
dot(mpB,4bp+blue);
dot(npA,4bp+red);
dot(npB,4bp+blue);

draw(pA--pA+(1,0),
grey,Arrow);
```



*draw(rotate(180+degrees((-1,0)))*scale(.5)*Label("$\vv{v}=(-1,0)$"),pB--pB+(-1,0),LeftSide, grey,Arrow);*
draw(mpA--mpA+(1,1),red,Arrow);
*draw(rotate(degrees((1,-1)))*scale(.5)*"$\vv{v}=(1,-1)$",mpB--mpB+(1,-1),red,Arrow);*
draw(npA--npA+(2,-1),blue,Arrow);
*draw(rotate(180+degrees((-1,-1)))*scale(.5)*"$\vv{v}=(-1,-1)$",npB--npB+(-1,-1), LeftSide,blue,Arrow);*
xaxis(Label("x",position=EndPoint, align=NE),
*Ticks(scale(.7)*Label(),NoZero,Size=.8mm, size=.4mm),Arrow);*
yaxis(Label("y",position=EndPoint, align=NE),
*Ticks(scale(.7)*Label(),NoZero,Size=.8mm, size=.4mm),Arrow);*
label("Cách dùng dirtime",truepoint(SE),NW);

- real **reltime(path p, real l)**;

returns the time on path p at the relative fraction l of its arclength.
trả lại thời điểm trên path p tại phân số tương đối l độ dài cung của nó.
* Sự so sánh arctime(path p, real L) và reltime(path p, real l)
Giống: Cả hai điều trả lại "thời điểm" path.
Khác:
L trong **arctime** có phạm vi tối thiểu từ **0** đến **arclength(p)**, còn l trong **reltime** có phạm vi tối thiểu là [0,1]. Dĩ nhiên, nếu vượt ra ngoài phạm vi này thì mặc định bị chuyển đổi thành tương ứng với điểm cuối của p.

- pair **relpoint(path p, real l)**;

returns the point on path p at the relative fraction l of its arclength.
trả lại điểm trên path p tại phân số tương đối l độ dài cung của nó.

- pair **midpoint(path p)**;

returns the point on path p at half of its arclength.
trả lại điểm trên path p tại nửa độ dài cung của nó

- path **reverse(path p)**;

returns a path running backwards along p.
trả lại một path chạy ngược hướng dọc theo p.

- path **subpath(path p, int a, int b)**;

returns the subpath of p running from node a to node b. If a > b, the direction of the subpath is reversed.
trả lại path con của p chạy từ node a đến node b theo hướng cũng từ node a đến node b. Nếu a>b, hướng của subpath bị đảo ngược.

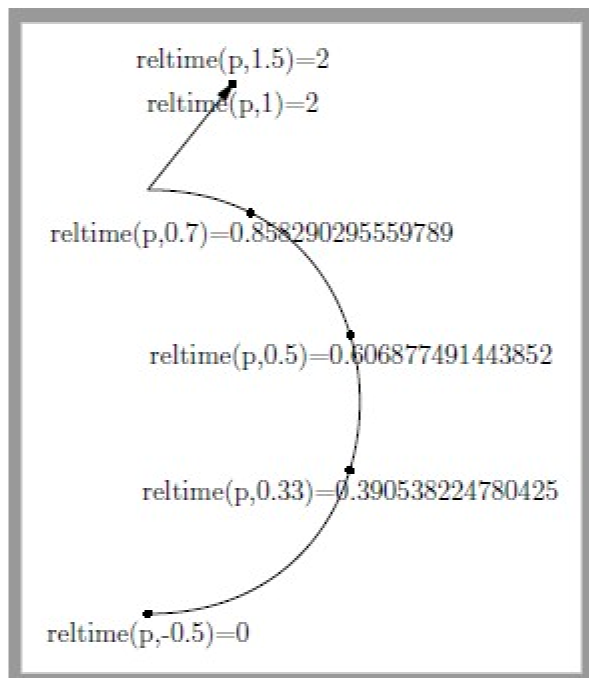
- path **subpath(path p, real a, real b)**;

returns the subpath of p running from path time a to path time b, in the sense of point(path, real). If a > b, the direction of the subpath is reversed.
trả lại path con của p chạy từ thời điểm path a đến thời điểm path b, theo nghĩa của point(path p, real t). Nếu a>b, hướng của subpath bị đảo ngược.
Ví dụ 11:
a.
size(10cm);
path p = (0,-1){E}..{W}(0,1)--(0.4,1.5);

```
draw(p,Arrow);
dot("midpoint(p)",midpoint(p));
draw(shift(1,0)*reverse(p),red+1bp,Arrow);
shipout(bbox(3mm,invisible));
```

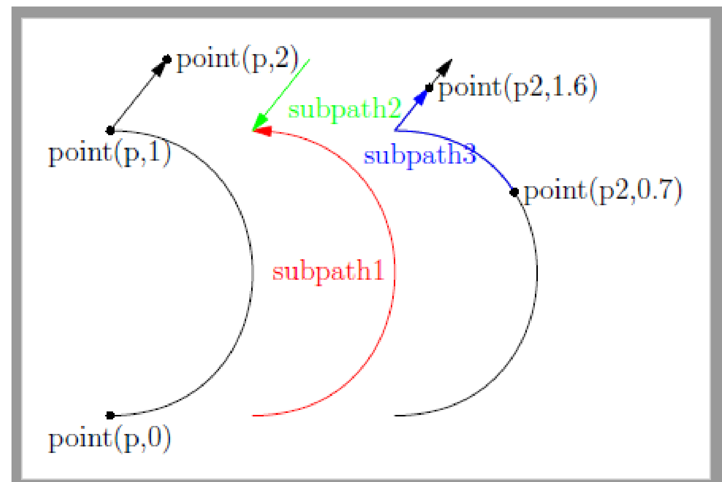
b.

```
size(10cm);
path p = (0,-1){E}..{W}(0,1)--(0.4,1.5);
draw(p,Arrow);
dot("reltime(p,0.33)"+"$=$"+(string) reltime(p,0.33),relpoint(p,0.33),dir(-90));
dot("reltime(p,0.7)"+"$=$"+(string) reltime(p,0.7),relpoint(p,0.7),dir(-90));
dot("reltime(p,1)"+"$=$"+(string) reltime(p,1),relpoint(p,1),dir(-90));
dot("reltime(p,-0.5)"+"$=$"+(string) reltime(p,-0.5),relpoint(p,-0.5),dir(-90));
dot("reltime(p,1.5)"+"$=$"+(string) reltime(p,1.5),relpoint(p,1.5),dir(90));
dot("reltime(p,0.5)"+"$=$"+(string) reltime(p,0.5),relpoint(p,0.5),dir(-90));
shipout(bbox(3mm,invisible));
```



c.

```
size(10cm);
path p = (0,-1){E}..{W}(0,1)--(0.4,1.5);
path p1=shift(-1,0)*p;
path p2=shift(1,0)*p;
draw(p1,Arrow);
draw(p2,Arrow);
dot("point(p,0)",point(p1,0),dir(-90));
dot("point(p,1)",point(p1,1),dir(-90));
dot("point(p,2)",point(p1,2));
path subpath1=subpath(p,0,1);
path subpath2=subpath(p,2,1);
path subpath3=subpath(p2,0.7,1.6);
```



```
draw("subpath1",subpath1,LeftSide,red,Arrow);
draw("subpath2",subpath2,LeftSide,green,Arrow);
draw("subpath3",subpath3,LeftSide,blue,Arrow);
dot("point(p2,0.7)",point(p2,0.7));
dot("point(p2,1.6)",point(p2,1.6));
shipout(bbox(3mm,invisible));
```

d. Ví dụ có chứa lệnh dir và relpoint (chỉnh sửa của ví dụ 118 file ASYfb.pdf)

```
unitsize(1cm);
pair[] A={(-3,0),(3,0)};
path p= A[0]{(1,-.5)}..{(.25,-1)}A[1];
draw(p,bp+blue);
int n=50;
pair[] B,C,D;
for(int i=0; i<=n;++i){
B[i]=relpoint(p,i/n);
C[i]=.5dir(p,i/n);
D[i]=shift(rotate(90)*C[i])*B[i];
draw(B[i]--D[i],Arrow(TeXHead));

}
draw(operator ..(... D),red);
shipout(bbox(5mm,invisible));
```

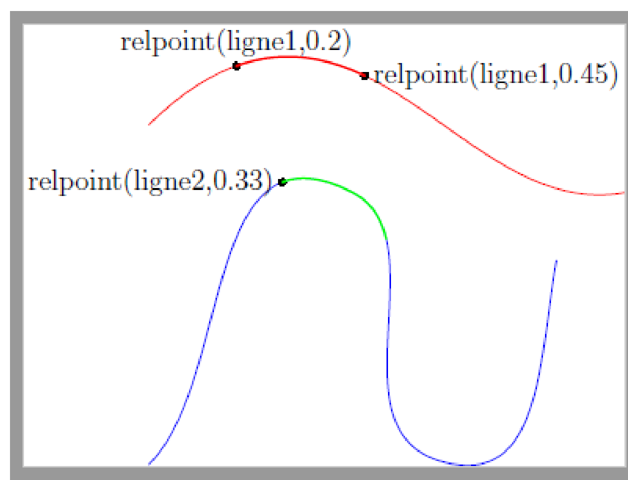
e.

Lấy một subpath bất kỳ trên **path p**, ta luôn lấy được một subpath trên **path g** sao cho độ dài cung của 2 subpath này bằng nhau.

```
unitsize(1cm);
path ligne1=(-3,1){dir(45)}..(-1,2)..{dir(10)}(4,0);
path ligne2=(-3,-1){dir(45)}..(0,3)..(1.5,-1)..{dir(80)}(3,2);
ligne2=shift(0,-3)*ligne2;

path subpath_ligne1=subpath(ligne1,reltime(ligne1,0.2),reltime(ligne1,0.45));
path testpath=subpath(ligne2,reltime(ligne2,0.33),reltime(ligne2,1));
path
subpath_ligne2=subpath(testpath,0,arctime(testpath,arclength(subpath_ligne1)));

draw(ligne1,red);
draw(ligne2,blue);
dot("relpoint(ligne1,0.2)",relpoint(ligne1,0.2),dir(90));
dot("relpoint(ligne1,0.45)",relpoint(ligne1,0.45));
dot("relpoint(ligne2,0.33)",relpoint(ligne2,0.33),dir(180));
draw(subpath_ligne1,red+1bp);
draw(subpath_ligne2,green+1bp);
```



- real[] **intersect(path p, path q, real fuzz=-1)**;

If p and q have at least one intersection point, return a real array of length 2 containing the times representing the respective path times along p and q, in the sense of point(path, real), for one such intersection point (as chosen by the algorithm described on page 137 of The MetaFontbook). The computations are performed to the absolute error specified by fuzz, or if fuzz < 0, to machine precision. If the paths do not intersect, return a real array of length 0.

nếu p và q có ít nhất một giao điểm (vị trí giao nhau đầu tiên), trả lại một mảng thực chiều dài 2 chứa các thời điểm biểu diễn các thời điểm path tương ứng chạy dọc p và q, theo nghĩa của point(path, real), cho giao điểm đó (như được chọn bởi thuật toán được mô tả ở trang 137 của file MetaFontbook). Sự tính toán được thực hiện theo sai số tuyệt đối được xác định bởi fuzz (tức là nếu giá trị fuzz càng nhỏ thì sự tính toán càng chính xác), hay nếu fuzz < 0, theo độ chính xác của máy. Nếu các path không giao nhau, trả lại một mảng chiều dài 0.

Bổ sung: Trong file MetaFontbook, ở trang 136, có câu:
Given two paths p and q, you can write
p intersectiontimes q
and the result will be a pair of times (t, u) such that point t of p ≈ point u of q.

Cho trước 2 path p và q, bạn có thể viết
p intersectiontimes q
và kết quả sẽ là một cặp thời điểm (t,u) sao cho **point(p,t) ≈ point(q,u)**.

- real[][] **intersections(path p, path q, real fuzz=-1)**;

Return all (unless there are infinitely many) intersection times of paths p and q as a sorted array of real arrays of length 2 (see [sort], page 73). The computations are performed to the absolute error specified by fuzz, or if fuzz < 0, to machine precision.

trả lại tất cả (ngoại trừ có vô hạn) các thời điểm giao nhau của các path p và q như **một mảng** được sắp xếp của **các mảng** thực có chiều dài bằng 2 (trong đó giá trị đầu tiên là thời điểm của path p, giá trị còn lại là thời điểm của path q). Sự tính toán được thực hiện theo sai số tuyệt đối được xác định bởi fuzz, hay nếu fuzz < 0, theo độ chính xác của máy.

- real[] **intersections(path p, explicit pair a, explicit pair b, real fuzz=-1)**;

Return all (unless there are infinitely many) intersection times of path p with the (infinite) line through points a and b as a sorted array. The intersections returned are guaranteed to be correct to within the absolute error specified by fuzz, or if fuzz < 0, to machine precision.

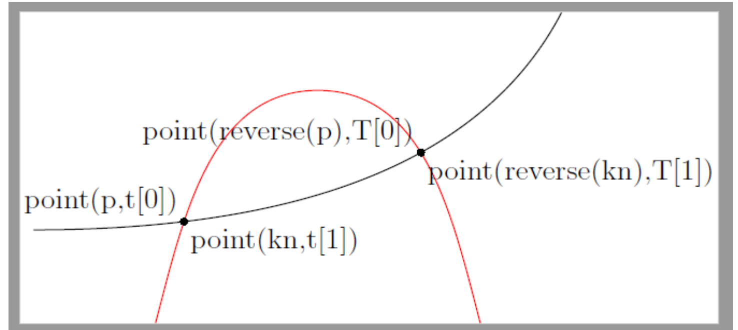
trả lại tất cả (ngoại trừ có vô hạn) các thời điểm giao nhau của path p với đường thẳng (vô hạn) đi qua các điểm a và b như một mảng được sắp xếp. Các sự giao nhau được trả lại được đảm bảo là đúng trong sai số tuyệt đối được xác định bởi fuzz, hay nếu fuzz < 0, theo độ chính xác của máy.

Ví dụ 12:

a.

```
size(10cm);
path p=(-3.5,.2){(1,0)}..{(1,2)}(3,3);
path kn=(-2,-1){(1,4)}..(0,2)..{(1,-4)}(2,-1);
draw(p);
draw(kn,red);

real[] t=intersect(p,kn,realEpsilon);
real[]
```

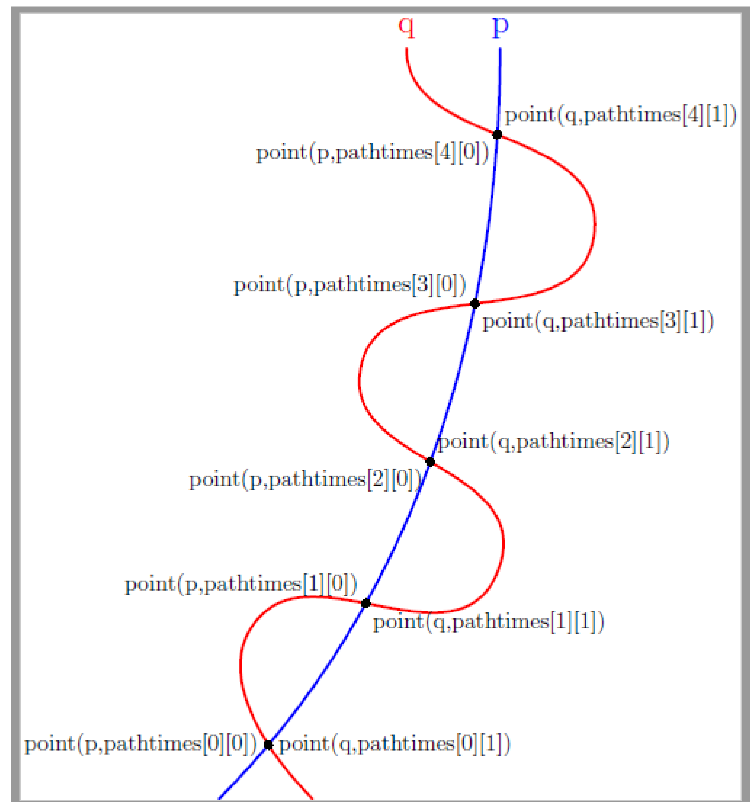



```
T=intersect(reverse(p),reverse(kn),realEpsilon);
// Ở đây tôi chọn giá trị fuzz là realEpsilon.
dot("point(p,t[0])",point(p,t[0]),dir(135));
label("point(kn,t[1])",point(kn,t[1]),dir(-45));
dot("point(reverse(p),T[0])",point(reverse(p),T[0]),dir(135));
label("point(reverse(kn),T[1])",point(reverse(kn),T[1]),dir(-45));
write(point(p,t[0]));  // Outputs: (-1.64998989132474,0.306526203263381)
write(point(kn,t[1])); // Outputs: (-1.64998989132474,0.30652620326338)
write(point(reverse(p),T[0]));  // Outputs: (1.26844917205345,1.19749256267475)
write(point(reverse(kn),T[1])); // Outputs: (1.26844917205345,1.19749256267475)
```

b.

```
size(10cm);
pair A=(0,-4), B=(-.5,-2), C=(2,-1.5), D=(.5,.5), E=(3,2), F=(1,4);
transform t=scale(.7);
path p =(-1,-4){(1,1)}..{(0,1)}(2,4);
path q =A{(-1,1)}..B..C..D..E..{(0,1)}F;
```

real[][]




```
pathtimes=intersections(p,q,realEpsilon);
draw(Label("p",Relative(1)),p,bp+blue);
draw(Label("q",Relative(1)),q,bp+red);
dot(t*"point(p,pathtimes[0][0])",
point(p,pathtimes[0][0]),dir(180));
label(t*"point(q,pathtimes[0][1])",
point(q,pathtimes[0][1]),dir(0));
dot(t*"point(p,pathtimes[1][0])",
point(p,pathtimes[1][0]),dir(135));
label(t*"point(q,pathtimes[1][1])",
point(q,pathtimes[1][1]),dir(-45));
dot(t*"point(p,pathtimes[2][0])",
point(p,pathtimes[2][0]),dir(-135));
label(t*"point(q,pathtimes[2][1])",
point(q,pathtimes[2][1]),dir(45));
dot(t*"point(p,pathtimes[3][0])",
point(p,pathtimes[3][0]),dir(135));
label(t*"point(q,pathtimes[3][1])",
point(q,pathtimes[3][1]),dir(-45));
dot(t*"point(p,pathtimes[4][0])",
point(p,pathtimes[4][0]),dir(-135));
label(t*"point(q,pathtimes[4][1])",
point(q,pathtimes[4][1]),dir(45));
// write(pathtimes);
```

c.

```
size(10cm);
pair A=(0,-4), B=(-.5,-2), C=(2,-1.5), D=(.5,.5), E=(3,2), F=(1,4);
```

```
path p =(-.5,-4)--(2,4);
path q =A{(-1,1)}..B..C..D..E..{(0,1)}F;
real[] pathtimes=intersections(q,(-0.5,-4),(2,4),realEpsilon);
draw(Label("p",Relative(1)),p,bp+blue);
draw(Label("q",Relative(1)),q,bp+red);
dot("point(q,pathtimes[0])",point(q,pathtimes[0]),dir(180));
dot("point(q,pathtimes[1])",point(q,pathtimes[1]),dir(135));
dot("point(q,pathtimes[2])",point(q,pathtimes[2]),dir(-135));
dot("point(q,pathtimes[3])",point(q,pathtimes[3]),dir(135));
dot("point(q,pathtimes[4])",point(q,pathtimes[4]),dir(-135));
```

d.

```
import patterns;

size(300);
path q=(-2,0)..(0,-1)..(2,0)..(0,1)..cycle;
pair A=relpoint(q,0.15),B=relpoint(q,0.3),
    C=relpoint(q,0.22),D=relpoint(q,0.75);

path curve1=A{dir(90)}.. tension 0.8 ..{dir(-90)}B;
path curve2=C{dir(120)}..{dir(135)}D;
real[] intersection_time=intersect(curve1,curve2,realEpsilon);

path part_1 = subpath(curve1,0,intersection_time[0])&
              subpath(curve2,intersection_time[1],reltime(curve2,1))&
                     subpath(q,reltime(q,0.75),reltime(q,1))&
                     subpath(q,reltime(q,0),reltime(q,0.15))&cycle;

add("hatch",shift(0,-3.6mm)*hatch(5mm,0.8bp+magenta));
add("fillhatch",hatch(5mm,8.5bp+green));
fill(part_1,pattern("fillhatch"));
fill(part_1,pattern("hatch"));
label("$\frac{1}{3}e$",(-0.45,0.5),UnFill);

path part_2 = subpath(curve1,0,intersection_time[0])&
              subpath(curve2,intersection_time[1],0)&
                     subpath(q,reltime(q,0.22),reltime(q,0.15))&cycle;

add("hatch1",shift(0,-3.6mm)*hatch(5mm,NW,0.8bp+orange));
add("fillhatch1",hatch(5mm,NW,8.5bp+green));
fill(part_2,pattern("fillhatch1"));
fill(part_2,pattern("hatch1"));
label(scale(.7)*"$0,25f$",(-0.6,-0.45),UnFill);

path part_3 = subpath(curve2,0,intersection_time[1])&
                 subpath(curve1,intersection_time[0],reltime(curve1,1))&
                 subpath(q,reltime(q,0.3),reltime(q,0.22))&cycle;

add("hatch2",shift(0,-3.6mm)*hatch(5mm,NW,0.8bp+orange));
add("fillhatch2",hatch(5mm,NW,8.5bp+blue));
fill(part_3,pattern("fillhatch2"));
fill(part_3,pattern("hatch2"));
```

```
label(scale(.7)*"$0,75f$",(0,-0.45),UnFill);

path part_4 = subpath(curve1,intersection_time[0],reltime(curve1,1))&
           subpath(q,reltime(q,0.3),reltime(q,0.75))&
                     subpath(curve2,reltime(curve2,1),intersection_time[1])&cycle;

add("hatch3",shift(0,-3.6mm)*hatch(5mm,0.8bp+magenta));
add("fillhatch3",hatch(5mm,8.5bp+blue));
fill(part_4,pattern("fillhatch3"));
fill(part_4,pattern("hatch3"));
label("$\frac{2}{3}e$",(0.45,0.5),UnFill);

draw(q,red);
draw(curve1);
draw(curve2);

draw(Label("$F$",Center,UnFill),(-0.6,-0.9){dir(-90)}..{dir(90)}(0,-0.9));
dot((-0.6,-0.9)^^(0,-0.9));
draw(Label("$\overline{F}$",Center,UnFill),(-0.3,0.9){dir(90)}..{dir(-90)}
(0.3,0.9));
dot((-0.3,0.9)^^(0.3,0.9));
draw(Label(minipage("\centering\scriptsize $T$\\ $0,3$ "),EndPoint),
   (-1.3,-0.68){dir(-170)}..{dir(-90)}((-1.3,-0.68)+0.4dir(-125)));
dot((-1.3,-0.68)^^(0.7,-0.75));
draw(Label(minipage("\centering\scriptsize $\overline{T}$\\ $0,7$"),EndPoint),
   (0.7,-0.75){dir(-10)}..{dir(-90)}((0.7,-0.75)+0.4dir(-70)));
draw(Label("$\Omega$",EndPoint),relpoint(q,0.52){dir(25)}..
{dir(0)}relpoint(q,0.52)+0.3dir(25));
```

Phân tích:

- Một path dù kín (cyclic) hay hở (non-cyclic) đều có 2 điểm cuối (end points), và path đó có hướng từ thời điểm path bằng **0** đến thời điểm path bằng **length(path p)**.

- Trong Asymptote:

"&" có tác dụng kết nối hai paths thành một path mới, liên tục (nghĩa là nó "kết nối hai điểm tiếp xúc thành một điểm");

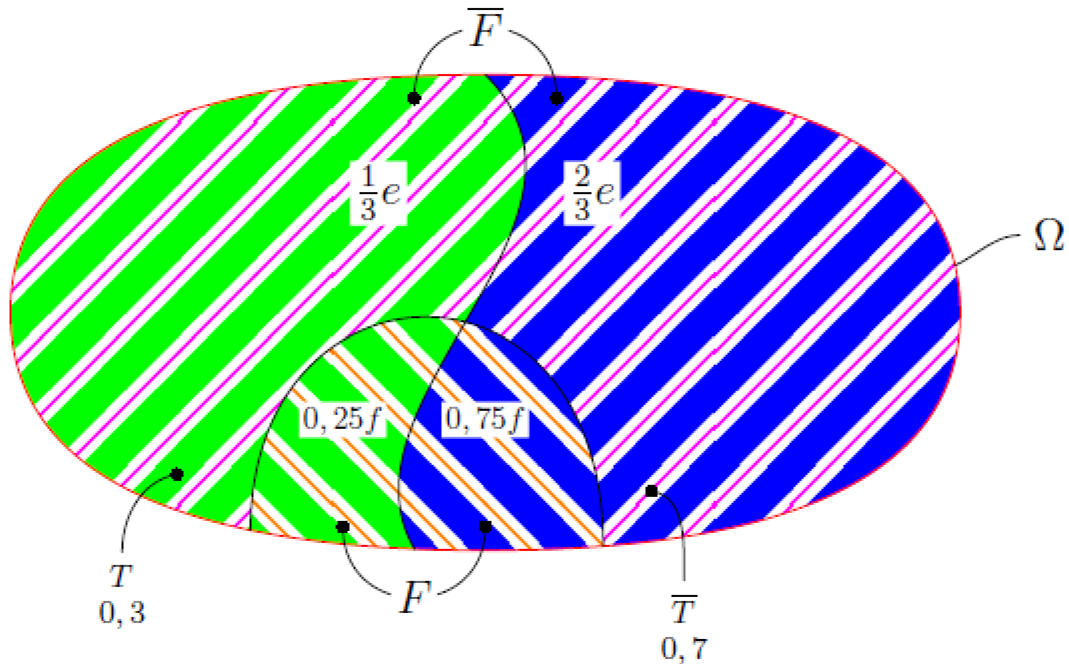
"&cycle" luôn nằm ở sau cùng, có tác dụng làm cho path **kín (cyclic)**.

Ví dụ minh họa cho "&cycle":

```
size(7cm);
path g=ellipse(0,3,1.5);
draw(g);
dot("A",relpoint(g,0.2),dir(45));
dot("B",relpoint(g,0.7),dir(-135));
path sub1=subpath(g,reltime(g,0.7),reltime(g,1));
path sub2=subpath(g,reltime(g,0),reltime(g,0.2));
path curve1=relpoint(g,0.7){dir(70)}.. tension 0.8 ..
{dir(120)}relpoint(g,0.2);
path combinedpaths3=sub1&sub2&reverse(curve1)&cycle;
filldraw(combinedpaths3,red+opacity(.8));
```

“&cycle” có tác dụng kết nối relpoint(sub1,0) và relpoint(reverse(curve1),1) thành 1 điểm để tạo path kín. Chú ý: vì **length(sub1) = 2** nên **length(combinedpaths3) = 4**.

- Ta tạo ra patterns bằng cách thêm hai lần pattern với độ rộng và vị trí



khác nhau.

- **real[] times(path p, real x, real fuzz=-1)**

returns all intersection times of path p with the vertical line through (x,0).
trả lại tất cả thời điểm giao nhau của path p với đường thẳng đứng đi qua (x,0).

- real[] **times(path p, explicit pair z, real fuzz=-1)**;

returns all intersection times of path p with the horizontal line through (0,z.y).
trả lại tất cả thời điểm giao nhau của path p với đường thẳng ngang đi qua (0,z.y).

Ví dụ 13:

a.

```
size(10cm);
import math;
pair A=(0,-4), B=(-.5,-2), C=(2,-1.5), D=(.5,.5), E=(3,2), F=(1,4);
path q =A{(-1,1)}..B..C..D..E..{(0,1)}F;
real a=-.25, b=2.5, c=1.33;
real[] ta=times(q,a,realEpsilon), tb=times(q,b,realEpsilon),
tc=times(q,c,realEpsilon);
draw(Label("q",Relative(1)),q,bp+red);
for(int k=0; k<tc.length; ++k) dot(point(q,tc[k]),3bp+green);
dot("point(q,ta[0])",point(q,ta[0]),dir(180));
dot("point(q,ta[1])",point(q,ta[1]),dir(135));
dot("point(q,tb[0])",point(q,tb[0]),dir(-40));
```

```
dot("point(q,tb[1])",point(q,tb[1]),dir(45));
draw(Label("$"+"x="+(string) a +"$",EndPoint,N),(a,-4)--(a,4),dashed+red);
draw(Label(format("$x=%f$",b),EndPoint,N),(b,-4)--(b,4),dashed+blue);
draw(Label(format("$x=%f$",c),BeginPoint,S),(c,-4)--(c,4),dashed+green);
write(ta.length);
write(tb.length);
write(tc.length);
```

b.

```
size(300);
path q=(0,0){dir(45)}..(1,0)..(1,.5)..(0,1)..cycle;
pair a=(0,.03), b=(0,.4), c=(0,.8);
real[]      ta=times(q,a,realEpsilon),      tb=times(q,b,realEpsilon),
tc=times(q,c,realEpsilon);
draw(q,bp+red);
for(int k=0; k<ta.length; ++k) dot(point(q,ta[k]),3bp+green);
for(int k=0; k<tb.length; ++k) dot(point(q,tb[k]),3bp+green);
for(int k=0; k<tc.length; ++k) dot(point(q,tc[k]),3bp+green);
draw(scale(.7)*Label("$"+"y="+(string) a.y+"$",EndPoint,N),
     (-1,a.y)--(1.5,a.y),dashed+red);
draw(scale(.7)*Label(format("$y=%f$",b.y),EndPoint,N),(-1,b.y)--
(1.5,b.y),dashed+blue);
draw(scale(.7)*Label(format("$y=%f$",c.y),EndPoint,N),(-1,c.y)--
(1.5,c.y),dashed+green);
```

- real[] **mintimes(path p)**;

returns an array of length 2 containing times at which path p reaches its minimal horizontal and vertical extents, respectively.
trả lại một mảng chiều dài 2 chứa các thời điểm ở tại đó path p đạt đến phạm vi ngang và dọc tối thiểu của nó, tương ứng.

- real[] **maxtimes(path p)**;

returns an array of length 2 containing times at which path p reaches its maximal horizontal and vertical extents, respectively.
trả lại một mảng chiều dài 2 chứa các thời điểm ở tại đó path p đạt đến phạm vi ngang và dọc tối đa của nó, tương ứng.

<u>Ví dụ 14:</u> Xem ví dụ 18.

- pair **intersectionpoint(path p, path q, real fuzz=-1)**;

returns the intersection point point(p,intersect(p,q,fuzz)[0]).
trả lại giao điểm **point(p,intersect(p,q,fuzz)[0])**.

- pair[] **intersectionpoints(path p, path q, real fuzz=-1)**;

returns an array containing all intersection points of the paths p and q.
trả lại một mảng chứa tất cả các điểm giao nhau của path p và q.

- pair **extension(pair P, pair Q, pair p, pair q)**;

returns the intersection point of the extensions of the line segments P--Q and p--q, or if the lines are parallel, (infinity,infinity).
trả lại giao điểm của các mở rộng của các đoạn thẳng P--Q và p--q hoặc nếu chúng song song, trả lại (infinity,infinity).

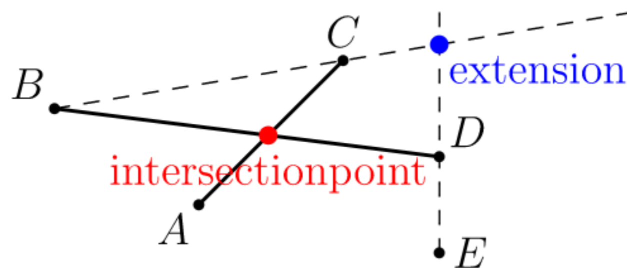
<u>Ví dụ 15:</u>

a.

```
size(7cm,0);
pair A=(0,0), B=(-1.5,1), C=(1.5,1.5), D=(2.5,.5), pE=(2.5,-.5);
path seg1=A--C, seg2=B--D;
pair
inter=intersectionpoint(seg1,seg2);
pair ext=extension(B,C,D,pE);

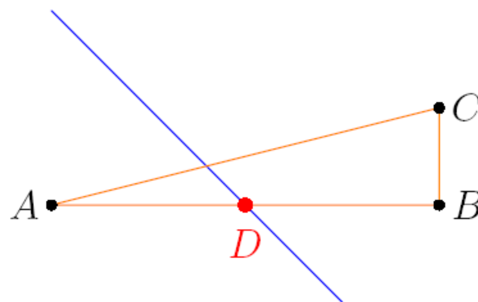
draw(seg1,linewidth(bp));
draw(seg2,linewidth(bp));
draw(B--interp(B,C,2),dashed);
draw(pE--interp(pE,D,2.5),dashed);


dot("intersectionpoint",inter,1.5*S,5bp+red);
dot("extension",ext,SE,5bp+blue);
dot("$A$",A,SW);dot("$B$",B,NW);
dot("$C$",C,N);dot("$D$",D,NE);
dot("$E$",pE,E);
shipout(bbox(5mm,white));
```



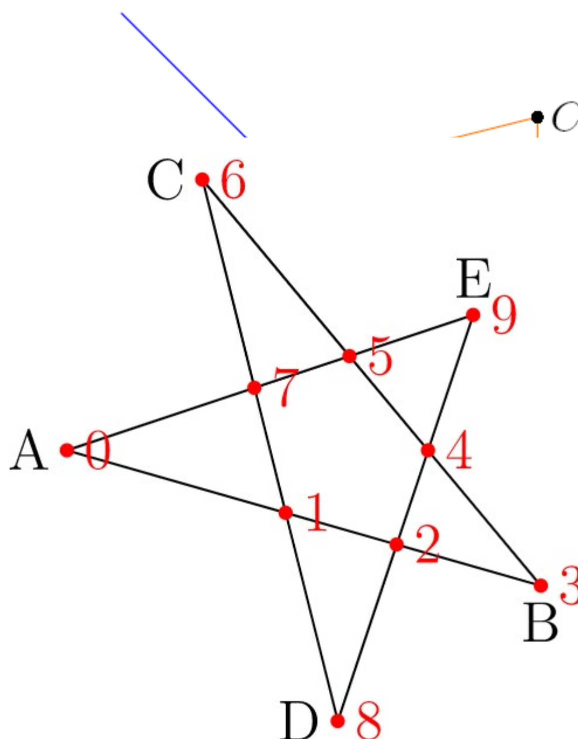
b.

```
unitsize(1cm);
path droite=(-2,2)--(1,-1);
path tri=(-2,0)--(2,0)--(2,1)--cycle;
pair pD=intersectionpoint(droite,tri);
draw(droite,blue);
draw(tri,orange);
dot("$A$",(-2,0),W);
dot("$B$",(2,0),E);
dot("$C$",(2,1),E);
dot("$D$",pD,2S,4bp+red);
shipout(bbox(5mm,white));
```



c.

```
unitsize(1cm);
path droite=(-2,2)--(1,-1);
path tri=(-2,0)--(2,0)--(2,1)--cycle;
pair
pD=intersectionpoint(droite,reverse
(tri));
draw(droite,blue);
draw(tri,orange);
dot("$A$",(-2,0),W);
dot("$B$",(2,0),E);
dot("$C$",(2,1),E);
dot("$D$",pD,2S,4bp+red);
shipout(bbox(5mm,white));
```



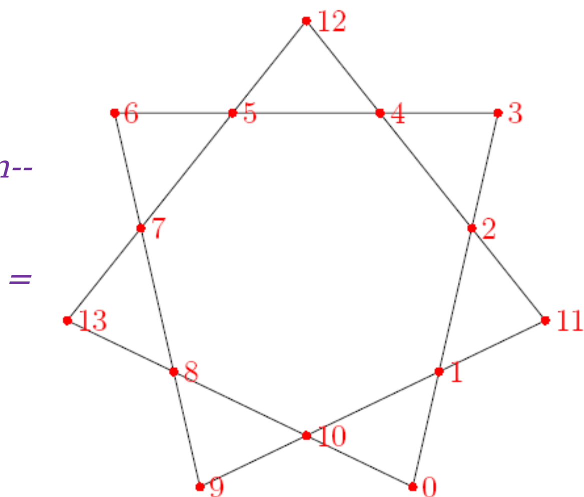
d.

```
unitsize(1cm);
pair A=(-1,0), B=(2.5,-1), C=(0, 2),
```

```
D=(1,-2), E=(2,1);
path c = A--B--C--D--E--cycle;
pair[] p = intersectionpoints(c,c);
draw(c);
for (int k=0; k<p.length; ++k){
    dot(format("%i",k),p[k],red);
}
label("A",A,dir(180));
label("B",B,dir(-90));
label("C",C,dir(180));
label("D",D,dir(180));
label("E",E,dir(90));
shipout(bbox(2mm,white));
```

e.

```
size(7.5cm,0);
path chemin;
for (int i=0; i<=7; ++i){
    chemin=chemin--point(polygon(7),2*i);
}
pair[] p = intersectionpoints(chemin,chemin);
draw(chemin);
for (int k=0; k<p.length; ++k){
    dot(format("%i",k),p[k],red);
}
shipout(bbox(2mm,white));
```



- slice **cut(path p, path knife, int n)**;

returns the portions of path p before and after the nth intersection of p with path knife as a structure slice (if no intersection exist is found, the entire path is considered to be 'before' the intersection):

```
struct slice {
  path before,after; }
```

The argument n is treated as modulo the number of intersections.

trả về các phần của path p trước và sau giao điểm thứ n của p với path knife bằng một cấu trúc slice (nếu không tồn tại sự giao nhau, toàn bộ path được xem xét là 'before' sự giao nhau):

struct slice { // cấu trúc slice có 2 members là before và after.
  path before,after;
}

Đối số n được xem như modulo số giao điểm.

- slice **firstcut(path p, path knife)**;

equivalent to cut(p,knife,0); Note that firstcut.after plays the role of the MetaPost cutbefore command.

Tương đương với cut(p,knife,0); Lưu ý rằng firstcut.after làm theo luật của lệnh MetaPost cutbefore.

- slice **lastcut(path p, path knife)**;

equivalent to cut(p,knife,-1); Note that lastcut.before plays the role of the MetaPost cutafter command.
Tương đương với cut(p,knife,-1); Lưu ý rằng lastcut.after làm theo luật của lệnh MetaPost cutafter.
*** Định nghĩa của 3 lệnh này trong plain_paths.asy như sau:

```
struct slice { // Khai báo cấu trúc slice
  path before,after;
}

slice cut(path p, path knife, int n)
{
  slice s; // khai báo slice s
  real[][] T=intersections(p,knife); // khai báo mảng 2 chiều T
  if(T.length == 0) {s.before=p; s.after=nullpath; return s;}
   T.cyclic=true; // khai báo lệnh này, ta mới có thể sử dụng giá trị -1 trong lệnh lastcut.
  real t=T[n][0]; // khai báo biến t là thời điểm trên path p ứng với giá trị n.
  s.before=subpath(p,0,t);
  s.after=subpath(p,t,length(p));
  return s;
}

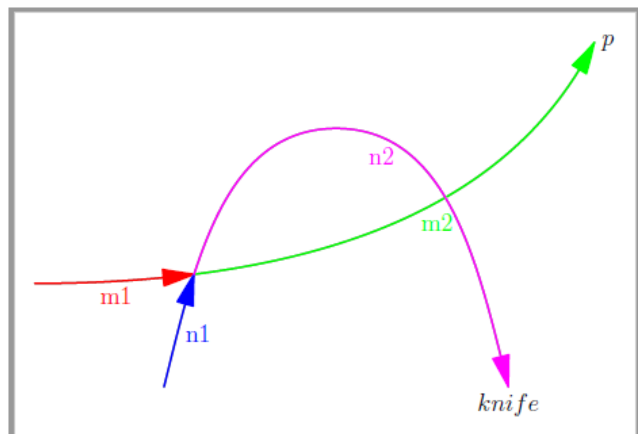
slice firstcut(path p, path knife)
{
  return cut(p,knife,0);
}

slice lastcut(path p, path knife)
{
  return cut(p,knife,-1);
}
```

Ví dụ 16:

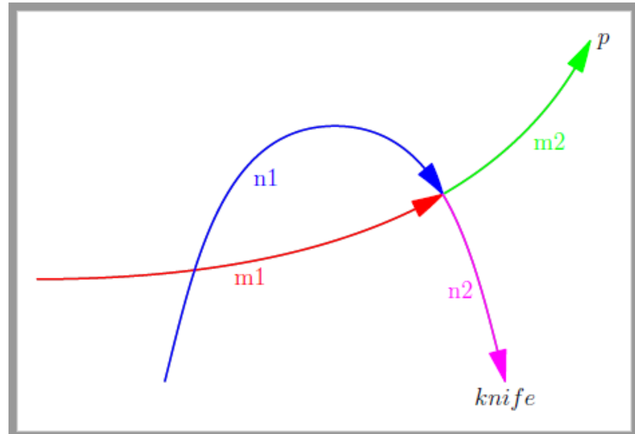
a.

```
size(300);
path p=(-3.5,.2){(1,0)}..{(1,2)}(3,3);
path kn=(-2,-1){(1,4)}..(0,2)..{(1,-4)}(2,-1);
draw(p,linewidth(1bp));
draw(kn,linewidth(1bp));
path m1=firstcut(p,kn).before, m2=firstcut(p,kn).after;
path n1=firstcut(kn,p).before, n2=firstcut(kn,p).after;
draw("m1",m1,1bp+red,Arrow);
draw("m2",m2,1bp+green,Arrow);
draw("n1",n1,1bp+blue,Arrow);
draw("n2",n2,1bp+magenta,Arrow);
label("$p$",relpoint(p,1),dir(0));
label("$knife$",relpoint(kn,1),dir(-90));
shipout(bbox(3mm,invisible));
```



b.

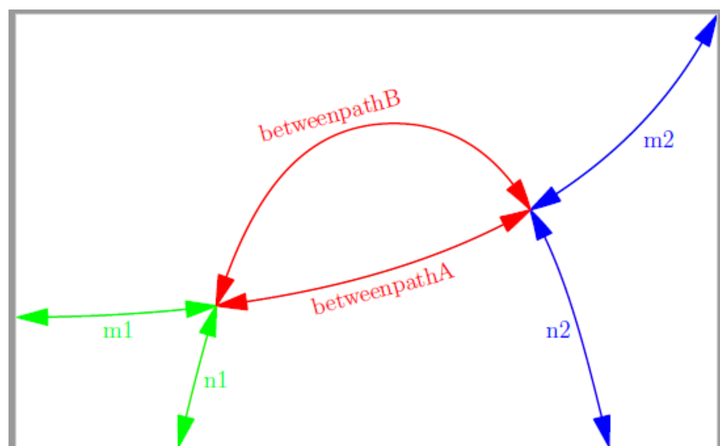
size(300);
path p=(-3.5,.2){(1,0)}..{(1,2)}(3,3);
path kn=(-2,-1){(1,4)}..(0,2)..{(1,-4)}(2,-1);
draw(p,linewidth(1bp));
draw(kn,linewidth(1bp));
path m1=lastcut(p,kn).before, m2=lastcut(p,kn).after;
path n1=lastcut(kn,p).before, n2=lastcut(kn,p).after;
draw("m1",m1,1bp+red,Arrow);



draw("m2",m2,1bp+green,Arrow);
draw("n1",n1,1bp+blue,Arrow);
draw("n2",n2,1bp+magenta,Arrow);
label("p",relpoint(p,1),dir(0));
label("$knife$",relpoint(kn,1),dir(-90));
shipout(bbox(3mm,invisible));

c.

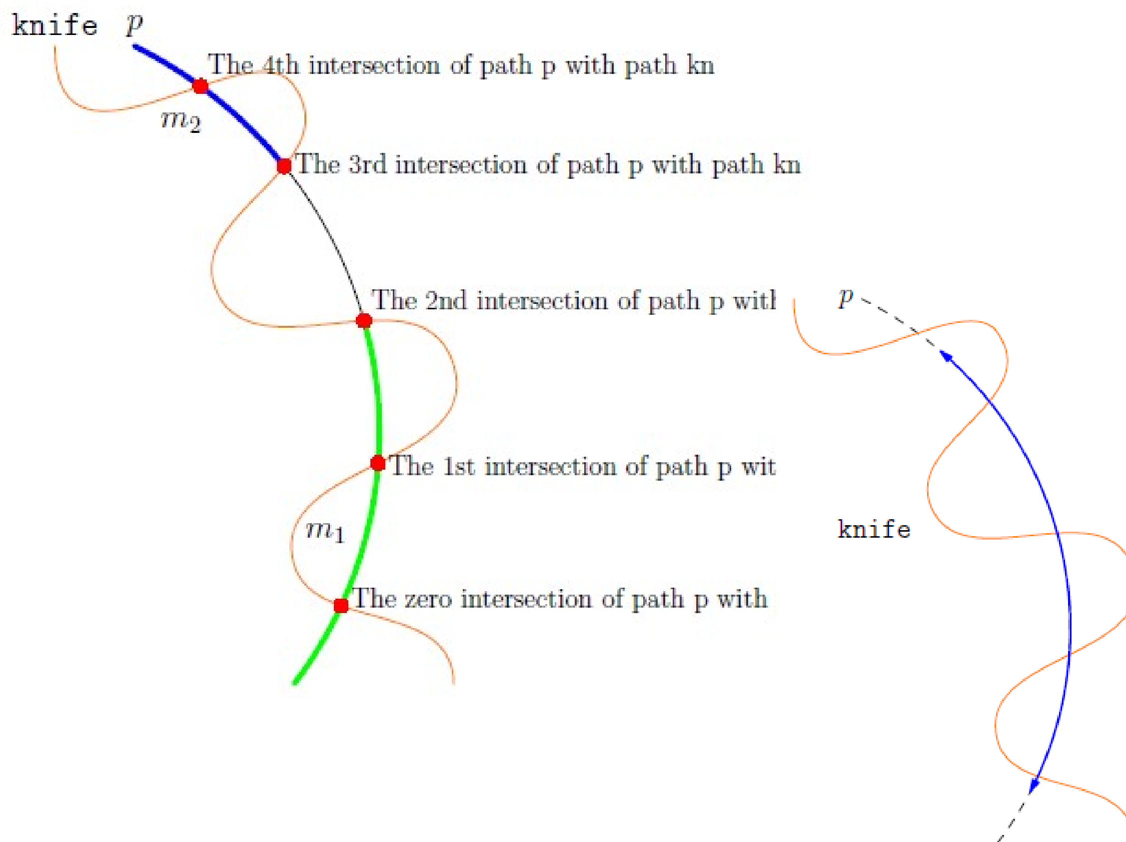
size(350);
path p=(-3.5,.2){(1,0)}..{(1,2)}(3,3);
path kn=(-2,-1){(1,4)}..(0,2)..{(1,-4)}(2,-1);
draw(p);
draw(kn,red);
real[][] A = intersections(p,kn,realEpsilon);
real[][] B = intersections(kn,p,realEpsilon);
path betweenpathA=subpath(p,A[0][0],A[1][0]);
path betweenpathB=subpath(kn,B[0][0],B[1][0]);
path m1=cut(p,kn,0).before;
path m2=cut(p,kn,1).after;
path n1=cut(kn,p,0).before;
path n2=cut(kn,p,1).after;
draw("m1",m1,bp+green,Arrows);
draw("m2",m2,bp+blue,Arrows);
draw(rotate(15)"betweenpathA",*
betweenpathA,bp+red,Arrows);



```
draw("n1",n1,bp+green,Arrows);
draw("n2",n2,bp+blue,Arrows);
draw(rotate(15)*"betweenpathB",
betweenpathB,LeftSide,bp+red,Arrows);
```

d.

```
unitsize(1cm);
path p=(2,0)..(3,4)..{(-2,1)}(0,8);
pair A=(4,0), B=(2,1.5), C=(4,4), D=(1,5), E=(2,7.5), F=(-1,8);
path kn=A{dir(90)}..B..C..D..E..{dir(90)}F;
real[][] pathtimes=intersections(p,kn,realEpsilon);
path m1=cut(p,kn,2).before;
path m2=cut(p,kn,3).after;
draw(p);
draw(kn,orange);
draw(m1,2bp+green);
draw(m2,2bp+blue);
dot(scale(.8)*Label("The zero intersection of path p with path kn",black),
point(p,pathtimes[0][0]),dir(15),5bp+red);
dot(scale(.8)*Label("The 1rd intersection of path p with path kn",black),
point(p,pathtimes[1][0]),dir(-15),5bp+red);
dot(scale(.8)*Label("The 2nd intersection of path p with path kn",black),
point(p,pathtimes[2][0]),dir(45),5bp+red);
dot(scale(.8)*Label("The 3rd intersection of path p with path kn",black),
point(p,pathtimes[3][0]),5bp+red);
dot(scale(.8)*Label("The 4th intersection of path p with path kn",black),
point(p,pathtimes[4][0]),dir(45),5bp+red);
label("$p$",endpoint(p),N);
label("\texttt{knife}",endpoint(kn),N);
label("$m_1$",point(m1,.5),2*dir(180));
label("$m_2$",point(m2,0.5),2*SW);
shipout(bbox(3mm,invisible));
```



e.

```
unitsize(1cm);
pair A=(4,0), B=(2,1.5), C=(4,4), D=(1,5),E=(2,7.5), F=(-1,8);
path p=(2,0)..(3,4)..{(-2,1)}(0,8);
path kn=A{dir(90)}..B..C..D..E..{dir(90)}F;
draw(p,dashed);
draw(kn,orange);
real[][] A = intersections(p,kn,realEpsilon);

path q=subpath(p,A[1][0]-0.5,A[3][0]+0.2);
draw(q,bp+blue,Arrows(5));
label("$p$",point(p,2),W);
label("\texttt{knife}",point(kn,3),3*SW);
shipout(bbox(3mm,white));
```

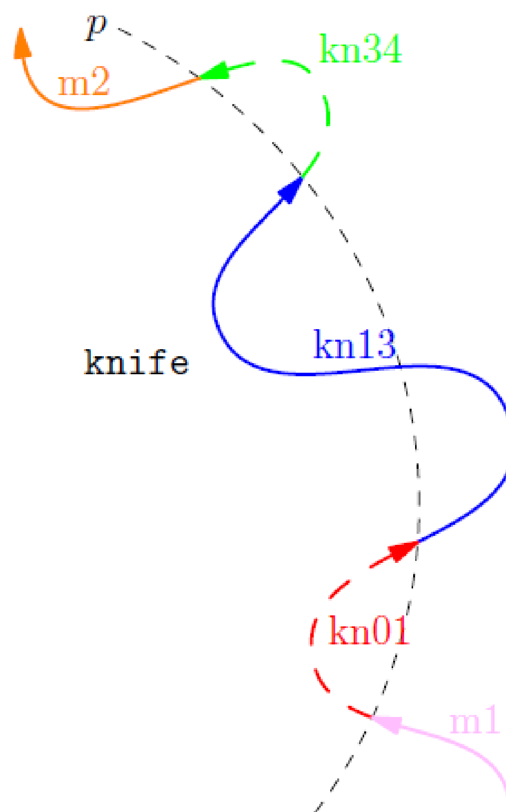
f.

```
unitsize(1cm);
pair A=(4,0), B=(2,1.5), C=(4,4), D=(1,5),
      E=(2,7.5), F=(-1,8);

path p=(2,0)..(3,4)..{(-2,1)}(0,8);
path kn=A{dir(90)}..B..C..D..E..{dir(90)}F;
draw(p,dashed);
//draw(kn,orange);
real[][] A = intersections(p,kn,realEpsilon);

path kn13=subpath(kn,A[1][1],A[3][1]);
path kn01=subpath(kn,A[0][1],A[1][1]);
path kn34=subpath(kn,A[3][1],A[4][1]);
path  m1=cut(kn,p,0).before,
         m2=cut(kn,p,4).after;

draw("kn13",kn13,1bp+blue,Arrow(8));
```




```
draw("kn01",kn01,1bp+red+dashed,Arrow(8));
draw("kn34",kn34,1bp+green+dashed,Arrow(8));
draw("m1",m1,1bp+pink,Arrow(8));
draw("m2",m2,1bp+orange,Arrow(8));

label("$p$",point(p,2),W);
label("\texttt{knife}",point(kn,3),3*SW);
shipout(bbox(3mm,white));
```

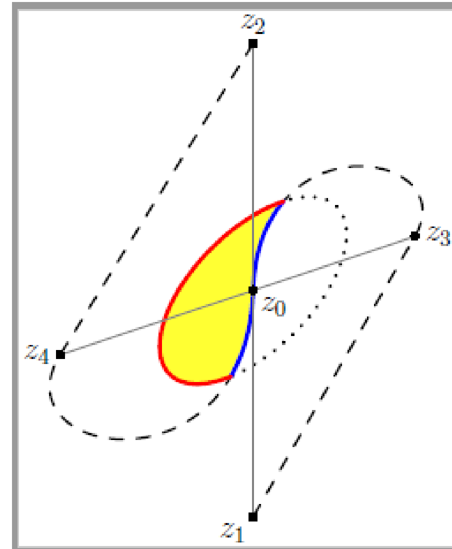
g.
// https://en.wikipedia.org/wiki/MetaPost, the third example.

// Đối với **whatever**, xem 8.20 MetaPost (tài liệu chính).

```
size(7cm);
pair z1=(0,0),z2=z1+2up;
pair z3=extension(z1,z1+10*dir(60),z2,z2+10*dir(-50));
pair z4=z3+(-1.5,-.5), z5=z1+dir(135);
pair z0=extension(z1,z2,z3,z4);

// fullcircle - Circle of diameter 1 centered on (0, 0)
path p0=shift(z0)*rotate(45)*yscale(0.5)*circle((0,0),0.5);
path p1=z2---z4..z0..z3---z1;
path p2=lastcut(firstcut(p1,p0).after,p0).before;
path p3=lastcut(firstcut(p0,p1).after,p1).before;

path p4=p2&p3&cycle;
fill(p4,rgb(1,1,0.2));
draw(p0,linetype(new real[] {0,3})+1.5bp);
draw(p1,linetype(new real[] {5,5})+bp);
draw(p2,blue+1.5bp);
draw(p3,red+1.5bp);
draw(z1--z2,.5white);
draw(z3--z4,.5white);
dot("$z_0$",z0,dir(-45));
dot("$z_1$",z1,dir(-135));
dot("$z_2$",z2,dir(90));
dot("$z_3$",z3);
dot("$z_4$",z4,dir(180));
```



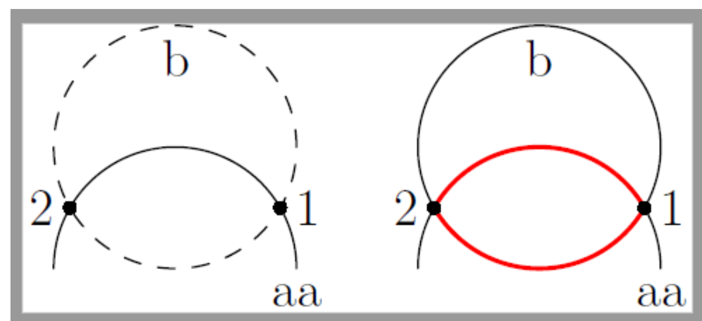
- path **buildcycle(... path[] p)**;

This returns the path surrounding a region bounded by a list of two or more consecutively intersecting paths, following the behaviour of the MetaPost buildcycle command.
Lệnh này trả lại path bao quanh một vùng bị giới hạn bởi một danh sách của 2 hay nhiều path giao nhau liên tiếp, theo cách hoạt động của lệnh MetaPost buildcycle.
Xem thêm ở file A User's Manual for MetaPost, 9.1 Building Cycles
TODO: phân tích luôn 9.1 Building Cycles.
Xem xét các ví dụ từ 137 đến 143 trong file ASYfb.pdf

Ví dụ 17:

a.

```
unitsize(1cm);
path aa=arc(0,1,0,180);
path b=circle((0,1),1);
pair[]
```




```
A=intersectionpoints(aa,b,realEpsilon);
transform t=shift(3,0);

draw(Label("aa",BeginPoint),aa);
```

```
draw(Label("b",Relative(0.25),LeftSide),b,dashed);
dot("$1$",A[0]);
dot("$2$",A[1],dir(180));

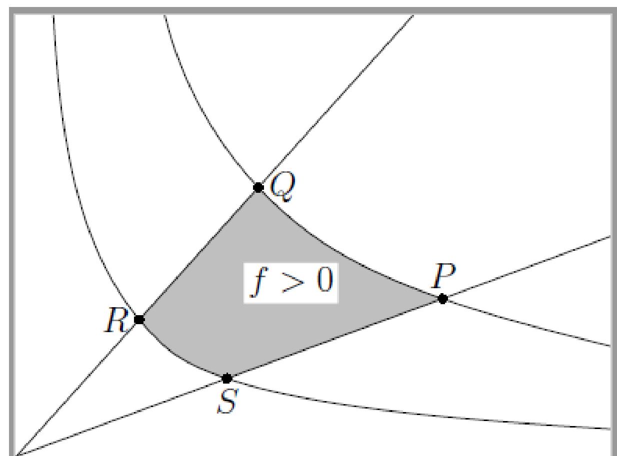
draw(Label("aa",BeginPoint),t*aa);
draw(Label("b",Relative(0.25),LeftSide),t*b);
draw(t*arc(0,A[0],A[1]),red+bp);
draw(t*arc((0,1),A[1],A[0]),red+bp);
dot("$1$",t*A[0]);
dot("$2$",t*A[1],dir(180));
```

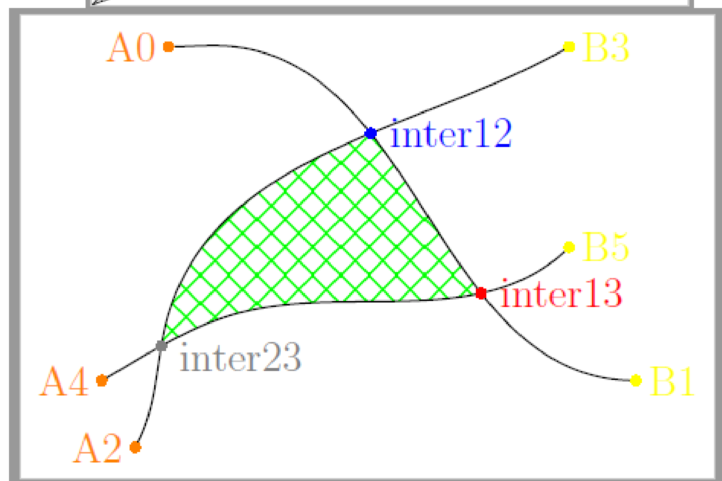
Xét thêm ví dụ này:

```
size(300);
import patterns;
add("name",hatch(2mm,red));
path g=circle(0,1);
path h=circle(sqrt(3),1);
//fill(buildcycle(g,h),pattern("name"));
fill(buildcycle(h,g),pattern("name"));
draw(g^^h);
```

Todo: đánh giá và phân tích sự khác nhau.

b. buildcycle.asy






```
size(200);
real h=2inch, w=2.7inch;

path[] p;
for(int k=0; k<2; ++k) {
  int i=2+2*k;
  int ii=i**2;
   p[k]=(w/ii,h){1,-ii}::(w/i,h/i)::(w,h/ii){ii,-1};
}

path q0=(0,0)--(w,0.5h);
path q1=(0,0)--(w/1.5,h);

path s=buildcycle(q0,p[0],q1,p[1]);
fill(s,mediumgrey);
draw(q0);
draw(p[0]);
draw(q1);
draw(p[1]);

dot("$P$",intersectionpoint(p[0],q0),N);
dot("$Q$",intersectionpoint(p[0],q1),E);
dot("$R$",intersectionpoint(p[1],q1),W);
dot("$S$",intersectionpoint(p[1],q0),S);
```

```
label("$f > 0$",0.5*(min(s)+max(s)),UnFill);
```

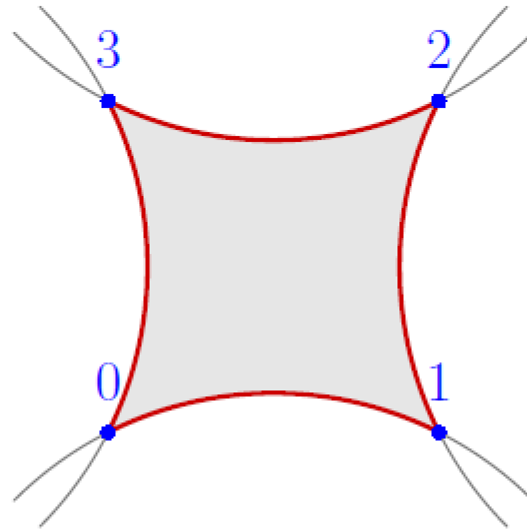
Todo: Nói về thứ tự các path trong buildcycle!

c.

```
import patterns;
size(7cm,0);
path line1=(-3,3){dir(0)}..(-2,3)..{dir(0)}(4,-2), line2=(-3.5,-3){dir(60)}..(-3,-1)..{dir(30)}(3,3),
     line3=(-4,-2){dir(30)}..(-2,-1)..{dir(45)}(3,0);
pair inter12=intersectionpoint(line1,line2), inter13=intersectionpoint(line1,line3),
     inter23=intersectionpoint(line2,line3);
add("AinterB",crosshatch(2mm,green));
fill(buildcycle(line1,line2,line3),pattern("AinterB"));
draw(line1^^line2^^line3);
pair[] dot={(-3,3),(4,-2),(-3.5,-3),(3,3),(-4,-2),(3,0)};
for (int i=0;i<=4;i+=2){ dot("A"+string(i),dot[i],W,orange);}
for (int i=1;i<=5;i+=2){ dot("B"+string(i),dot[i],E,yellow);}
dot("inter12",inter12,1.5*dir(0),blue);
dot("inter13",inter13,1.5*dir(0),red);
dot("inter23",inter23,1.5*dir(-20),gray);
shipout(bbox(1mm,white));
```

d.

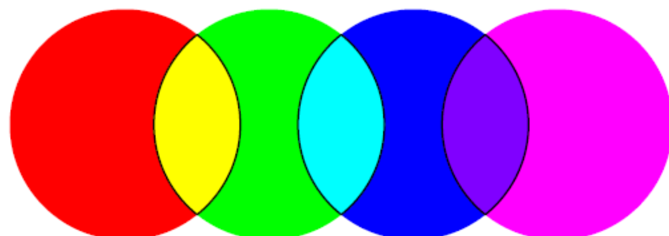
```
unitsize(1cm);
path [] c; path [] h;
c[1] = (-2,0){(1,1)}..{(1,-1)}(2,0);
h[1] = shift(0,-1.8)*c[1];
h[2] = shift(1.8,0)*rotate(90)*c[1];
h[3] = shift(0,1.8)*rotate(180)*c[1];
h[4] = shift(-1.8,0)*rotate(-90)*c[1];
```




```
draw(h[1]^^h[2]^^h[3]^^h[4],grey);
c[5] = buildcycle(h[1],h[2],h[3],h[4]);
fill(c[5], lightgrey);
draw(c[5], .8red+1bp);
for(int i=0; i<length(c[5]); ++i){
dot(string(i),point(c[5],i),2*N, blue+3bp);
}
shipout(bbox(1mm,white));
```

e.

```
unitsize(1cm);
```



```
pen[] p={red,green,blue,magenta};
path [] s;
for(int i=0; i<=3;++i){
  s[i]=shift(1.25*i,0)*unitcircle;
  fill(s[i],p[i]);
}
for(int i=0; i<=2;++i){
  filldraw(buildcycle(s[i+1],s[i]),p[i]+p[i+1]);
}
shipout(bbox(3mm,white));
```

- pair **min(path p)**;

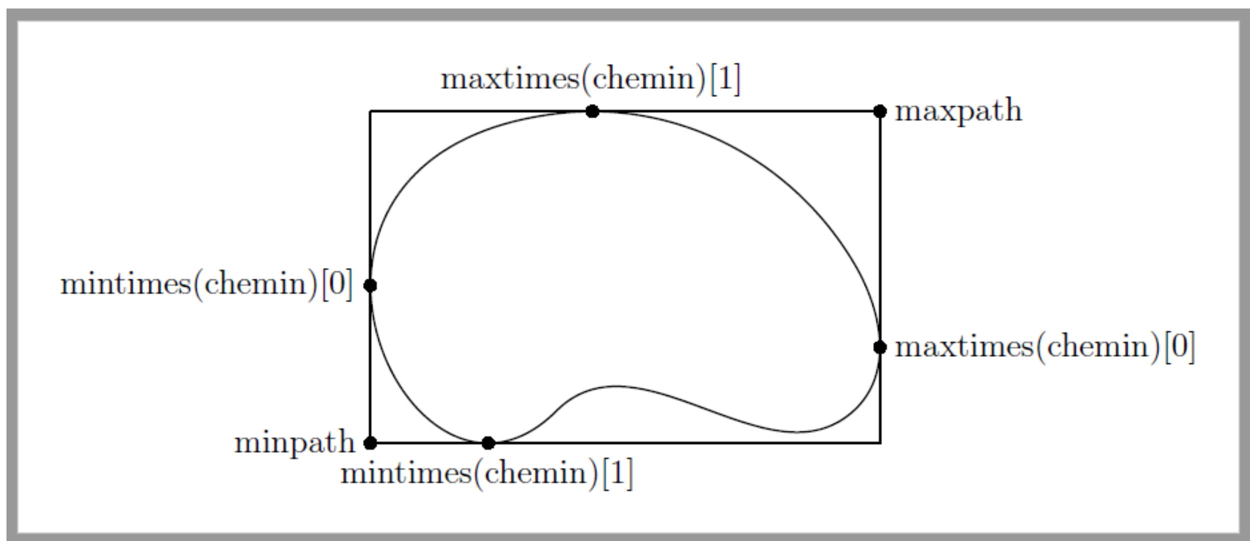
returns the pair (left,bottom) for the path bounding box of path p.
trả lại tọa độ (left,bottom) của path hộp bao quanh của path p.

- pair **max(path p)**;

returns the pair (right,top) for the path bounding box of path p.
trả lại tọa độ (right,top) của path hộp bao quanh của path p.

Ví dụ 18:

```
size(300);
path chemin=(0,0){dir(45)}..(1,0)..(1,.5)..(0,1)..cycle;
draw(chemin);
real[] a=mintimes(chemin),b=maxtimes(chemin);
pair minpath =min(chemin);
pair maxpath =max(chemin);
dot(scale(.7)*"mintimes(chemin)[0]",point(chemin,a[0]),dir(180));
dot(scale(.7)*"mintimes(chemin)[1]",point(chemin,a[1]),dir(-90));
dot(scale(.7)*"maxtimes(chemin)[0]",point(chemin,b[0]),dir(0));
dot(scale(.7)*"maxtimes(chemin)[1]",point(chemin,b[1]),dir(90));
draw(box(min_chemin,max_chemin));
dot(scale(.7)*"minpath", minpath,dir(180));
dot(scale(.7)*"maxpath", maxpath);
shipout(bbox(3mm,white));
```



Phân tích:

a[0] hay mintimes(chemin)[0] chính là thời điểm ngang tối thiểu của path chemin.
a[1] hay mintimes(chemin)[1] chính là thời điểm dọc tối thiểu của path chemin.
b[0] hay maxtimes(chemin)[0] chính là thời điểm ngang tối đa của path chemin.
b[1] hay maxtimes(chemin)[1] chính là thời điểm dọc tối đa của path chemin.
minpath có tọa độ là (point(chemin,a[0]).x, point(chemin,a[1]).y).
maxpath có tọa độ là (point(chemin,b[0]).x, point(chemin,b[1]).y).

- int **windingnumber(path p, pair z)**;

returns the winding number of the cyclic path p relative to the point z. The winding number is positive if the path encircles z in the counterclockwise direction. If z lies on p the constant undefined (defined to be the largest odd integer) is returned.
Tham khảo: Winding number
Xem sách Visual Complex Functions_ An Introduction with Phase Portraits.

- bool **interior(int windingnumber, pen fillrule)**;

returns true if windingnumber corresponds to an interior point according to fillrule.

Ví dụ 19:
https://tex.stackexchange.com/a/520622/219419
https://tex.stackexchange.com/q/419869/219419
https://tex.stackexchange.com/a/405241/219419

- bool **inside(path p, pair z, pen fillrule=currentpen)**;

returns true iff the point z lies inside or on the edge of the region bounded by the cyclic path p according to the fill rule fillrule (see [fillrule], page 42).
trả lại true nếu và chỉ nếu điểm z nằm bên trong hay trên rìa của vùng bị bao quanh bởi path kín p theo luật fill fillrule.

- int **inside(path p, path q, pen fillrule=currentpen)**;

returns 1 if the cyclic path p strictly contains q according to the fill rule fillrule (see [fillrule], page 42), -1 if the cyclic path q strictly contains p, and 0 otherwise.
trả lại 1 nếu path kín p chứa hoàn toàn q theo luật fill fillrule, -1 nếu path kín q chứa hoàn toàn p, và 0 trong trường hợp khác.

- pair **inside(path p, pen fillrule=currentpen)**;

returns an arbitrary point strictly inside a cyclic path p according to the fill rule fillrule (see [fillrule], page 42).
trả lại một điểm bất kỳ bên trong hoàn toàn một path kín p theo luật fill fillrule.

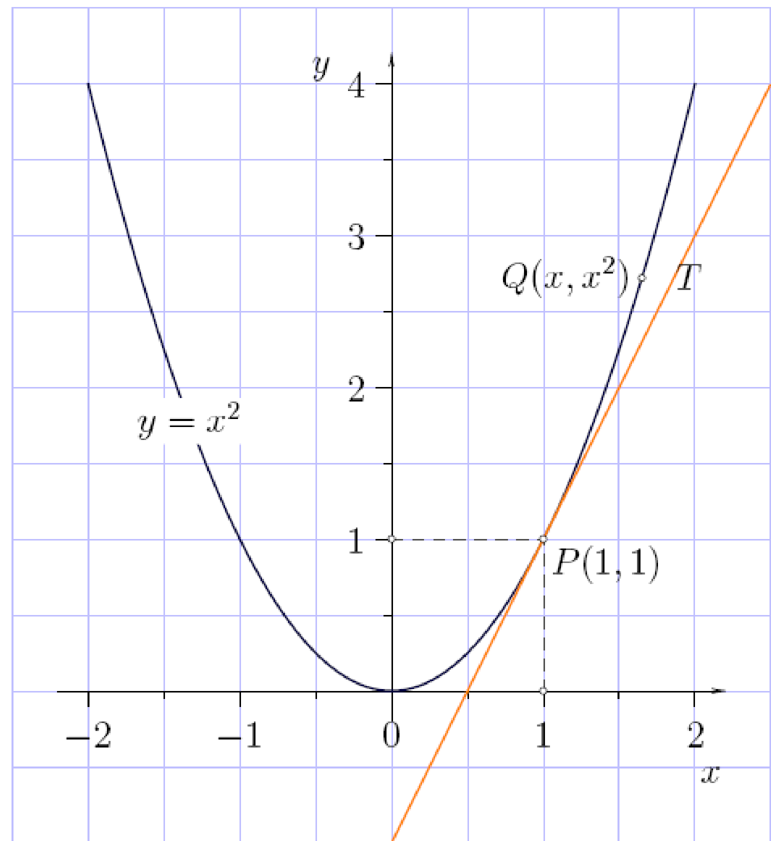
Ví dụ 20:

- path[] **strokepath(path g, pen p=currentpen)**;

returns the path array that PostScript would fill in drawing path g with pen p.
trả lại mảng path mà PostScript sẽ fill

Ví dụ 21:
https://tex.stackexchange.com/a/127785/219419

Ví dụ 3: this link




```
import graph;
import math;
unitsize(5cm);
size(300,0);
import fontsize;
defaultpen(fontsize(14pt));
texpreamble(
"\usepackage{lmodern}"
+"\usepackage{amsmath}"
+"\usepackage{amsfonts}"
+"\usepackage{amssymb}"
);

real xmin=-2,xmax=-xmin;
real ymin=0,ymax=4;
real dxmin=0.2;
real dxmax=dxmin;
real dymin=dxmin;
real dymax=dxmax;

add(shift(-2.5,-
1)*scale(0.5)*grid(10,11,paleblue+0.3bp));

xaxis("$x$",xmin-dxmin,xmax+dxmax,RightTicks(Step=1,step=0.5),
      Arrow(HookHead,size=2),above=true);
yaxis("$y$",ymin-dymin,ymax+dymax,
LeftTicks
(Step=1,step=0.5,OmitTick(0)),Arrow(HookHead,size=2),above=true);

real f(real x){return x^2;}
```

```
guide gf=graph(f,xmin,xmax,operator..);

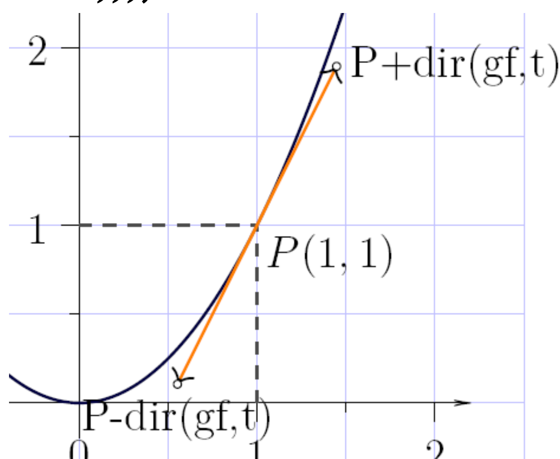
real x=1, t=times(gf,x)[0];
pair P=point(gf,t), Q=relpoint(gf,6/7);

draw(gf,darkblue+0.9bp);
draw((P.x,0)--P--(0,P.y),gray(0.3)+0.8bp+linetype(new real[]{5,5})
+linecap(0));
drawline(P,P+dir(gf,t),orange+0.9bp);
dot((P.x,0)^^P^^(0,P.y)^^Q,UnFill);
label("$y=x^2$",relpoint(gf,1/4),UnFill);
label("$P(1,"+string(round(P.y))+")$",P,plain.SE);
label("$Q(x,x^2)$",Q,plain.W);
label("$T$",Q,3*plain.E);

shipout(bbox(3mm,invisible));
```

Phân tích:
Ở đây, ta chỉ phân tích những thứ liên quan đến **path** thôi.
- Function **drawline** (part of the basic Asymptote module math.asy), allows to draw the visible portion of the (infinite) line going through two points.
- Để cho dễ hiểu, tôi sẽ cho bạn các lệnh sau

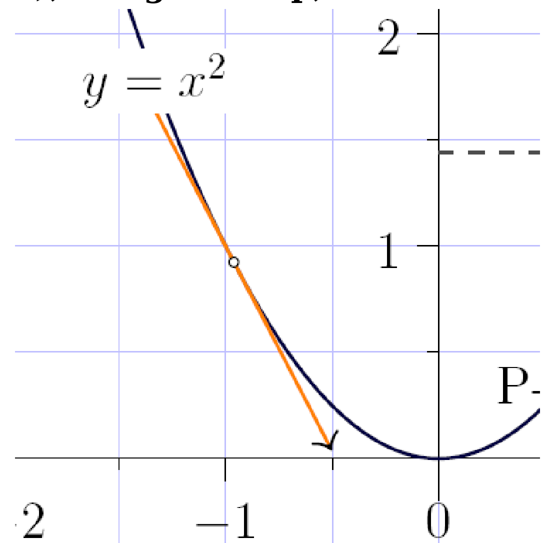
draw(P-dir(gf,t)--P+dir(gf,t), orange+0.9bp,Arrows(TeXHead,Fill(black)));
label("P-dir(gf,t)",P-dir(gf,t),S);
label("P+dir(gf,t)",P+dir(gf,t),E);
dot(P-dir(gf,t)^^P+dir(gf,t),UnFill);



* P-dir(gf,t)--P+dir(gf,t) cái này để vẽ đường thẳng tiếp tuyến với đồ thị gf tại điểm P. dir(gf,t) tức là “vecto pháp tuyến tại thời điểm t trên đồ thị gf.

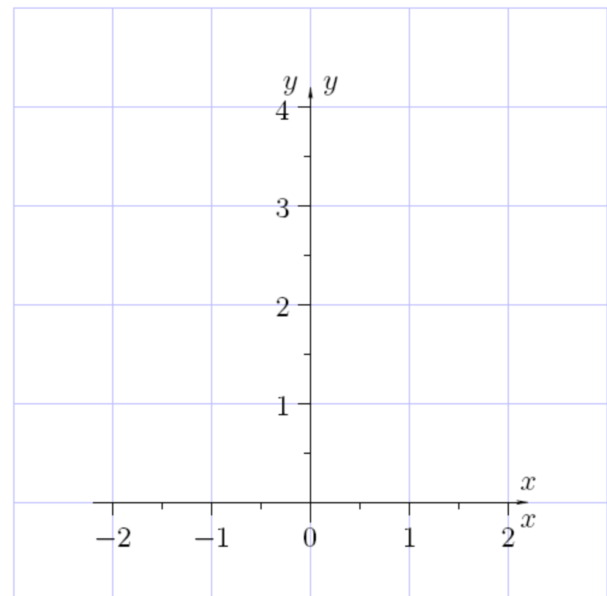
- Với lệnh **t=times(gf,x)[0];**, ta có thể xác định tiếp tuyến của đồ thị theo **hoành độ**. Một cách khác để vẽ tiếp tuyến tại một điểm trên đồ thị nhưng **chỉ liên quan đến độ dài cung** như sau:

real bbn=reltime(gf,0.35); // tác dụng như t=times(gf,x)[0];
pair bbnn=relpoint(gf,0.35); // tác dụng như P=point(gf,t);
draw(bbnn-dir(gf,bbn)--bbnn+dir(gf,bbn),orange+0.9bp,
Arrows(TeXHead,Fill(black)));



Bổ sung thêm:

*//add(shift(-3,-1)*scale(0.5)*grid(12,12,paleblue+0.3bp));*
*add(shift(-3,-1)*grid(6,6,paleblue+0.3bp));*
xaxis(xmin-dxmin,xmax+dxmax,RightTicks(Step=1,step=0.5),Arrow(HookHead,size=2),above=true);
*label("x",(xmax,0),2*dir(45));*
*label("x",(xmax,0),2*dir(-45));*
yaxis(ymin-dymin,ymax+dymax,LeftTicks(Step=1,step=0.5,OmitTick(0)),Arrow(HookHead,size=2),above=true);
*label("y",(0,ymax),2*dir(45));*
*label("y",(0,ymax),2*dir(135));*



- //add(shift(-3,-1)*scale(0.5)*grid(12,12,paleblue+0.3bp));
add(shift(-3,-1)*grid(6,6,paleblue+0.3bp));

Thêm nền lưới (chú ý tùy chỉnh vị trí thích hợp để đặt nền lưới, chú ý khi nào dùng **scale**(tăng số ô theo chiều hoành độ, theo chiều tung độ cho phù hợp), tự thử nhé.

- label("x",(xmax,0),2*dir(45));
label("x",(xmax,0),2*dir(-45));

đặt nhãn theo vị trí thích hợp, nên dùng để tùy chỉnh chứ nếu để nhãn luôn trong lệnh xaxis(yaxis), sẽ không đẹp.

Ví dụ 4: được vẽ từ link này
Ví dụ này để cập đến **real arctime(path p, real L);** và **pair arcpoint(path p, real L);**
Khái niệm trong file metapost:
If p is a path and a is a number between 0 and arclength p, **arctime a of p** gives the time t such that **arclength subpath (0,t) of p = a**.

Trả lại thời điểm t sao cho độ dài cung con từ 0 đến t của p bằng a. Ở đây L=a.

```
import graph;
import math;
size(500);
defaultpen(linewidth(1.2bp));
defaultpen(fontsize(20pt));

add(shift(-3,-1)*scale(0.5)*grid(12,12,paleblue+0.3bp));

pair I=(1,1);
drawline(I,I+dir(90),green);

label("I",I,SE);
path circ=shift(0,1)*scale(1)*unitcircle;
draw(circ);
draw((-1.2,1)--(1.2,1)^^(0,-0.2)--(0,2.2));

pair O=(0,1),J=(0,2);
pair M=point(circ,arctime(circ,pi/5+pi/6)),Mmap=I+(0,pi/5+pi/6);

label("$O$",O,SW);
label("$J$",J,SW);

draw(O--M);

path trajet;
trajet = subpath(circ,0,arctime(circ,pi/5+pi/6))..{left}(-3,2.5);
pair Pi=point(trajet,arctime(trajet,pi));
draw(trajet,0.5*green);


dot("$M$",M,2*dir(100));

path ga=I+(0,pi){left}..Pi;
path gb=Mmap{left}..M;
draw(ga,Arrow,Margin(2,2));
draw(gb,Arrow,Margin(1,1));
```

```
void addtick(picture pic=currentpicture, Label L, pair z, pair dir=E, pen p=currentpen)
// % https://melusine.eu.org/syracuse/asymptote/animations/a20/fig0160.asy
{
transform R=rotate(degrees(dir));
real width=1.5mm;
Label L=L.copy();
L.position(z);
L.align(NoAlign,E);
L.align.dir=R*L.align.dir*1.3*width/mm;
L.p(p);
pic.add(new void(frame f, transform t) {
path g=(-width,0)--(width,0);
picture opic;
draw(opic,shift(t*z)*R*g,p);
add(f,opic.fit());
});
add(pic,L);
}
addtick(Label("$\alpha$",align=E),Mmap);
addtick(Label("$\frac{\pi}{2}$",align=2E),I+(0,pi/2));
addtick(Label("$\pi$",align=E),I+(0,pi));
addtick(Label("-$\frac{\pi}{2}$",align=E),I+(0,-pi/2));
addtick(Label("$\pi$",align=NE),Pi,dir=dir(80));
addtick(Label("$1$",align=2E),I+dir(90));
shipout(bbox(3mm,invisible));
```

Guide

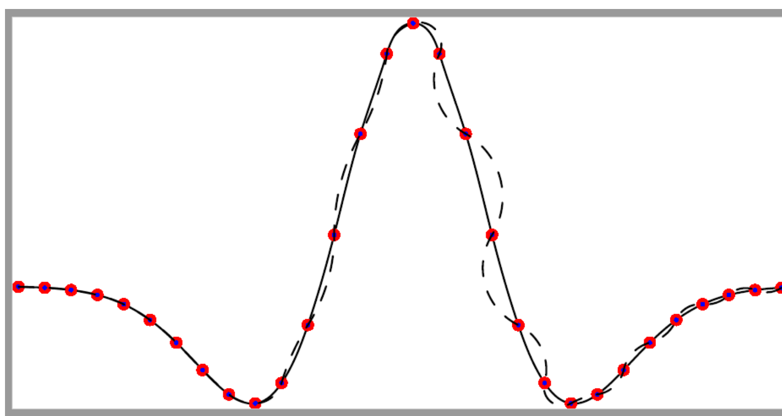
https://asymptote.sourceforge.io/FAQ/section7.html#pathguide

Định nghĩa trong tài liệu chính: Nó là an unresolved cubic spline (list of cubic-spline nodes and control points).

Sự khởi tạo ngầm cho nó là nullpath; .

Một gude thì tương tự như path ngoại trừ rằng sự tính toán cubic spline bị hoãn lại cho đến khi vẽ time (khi nào nó được phân giải thành một path); điều này cho phép hai guides với các điều kiện điểm cuối tự do được gắn lại cùng nhau một cách mịn, nhưng chỉ bị phân giải tại thời điểm đó; đường cong nét đứt được phân giải từng bước ở mỗi lần lặp lại, trước lúc toàn bộ tập các nodes được biết.

```
size(200);
real mexican(real x) {return (1-8x^2)*exp(-(4x^2));}
int n=30;
real a=1.5;
real width=2a/n;
guide hat;
path solved;
for(int i=0; i < n; ++i) {
real t=-a+i*width;
pair z=(t,mexican(t));
hat=hat..z;
solved=solved..z;
}
draw(hat);
dot(hat,red+3bp);
dot(solved,blue+1bp);
draw(solved,dashed);
```



Ví dụ sau cho biết khả năng của path và guide:

```
guide g;
for(int i=0; i < 10; ++i)
g=g--(i,i);
path p=g;
```

runs ***in linear time***, whereas

```
path p;
for(int i=0; i < 10; ++i)
p=p--(i,i);
```

runs ***in quadratic time***.

Nên phân biệt trước path và guide, rồi sau đó mới phân tích guide.

Linear time thì tốt hơn quadratic time.

Các lệnh con liên quan đến guide:

int size(guide g);

Analogous to **size(path p).**

int length(guide g);

Analogous to **length(path p)**.

bool cyclic(guide p);

Analogous to **cyclic(path p)**.

pair point(guide g, int t);

Analogous to **point(path p, int t)**.

guide reverse(guide g);

Analogous to reverse(path p). If g is cyclic and also contains a secondary cycle, it is first solved to a path, then reversed. If g is not cyclic but contains an internal cycle, only the internal cycle is solved before reversal. If there are no internal cycles, the guide is reversed but not solved to a path.

pair[] dirSpecifier(guide g, int i);

This returns a pair array of length 2 containing the outgoing (in element 0) and incoming (in element 1) direction specifiers (or (0,0) if none specified) for the segment of guide g between nodes i and i+1.

pair[] controlSpecifier(guide g, int i);

If the segment of guide g between nodes i and i+1 has explicit outgoing and incoming control points, they are returned as elements 0 and 1, respectively, of a two-element array. Otherwise, an empty array is returned.

tensionSpecifier tensionSpecifier(guide g, int i);

This returns the tension specifier for the segment of guide g between nodes i and i+1. The individual components of the tensionSpecifier type can be accessed as the virtual members in, out, and atLeast.

real[] curlSpecifier(guide g);

This returns an array containing the initial curl specifier (in element 0) and final curl specifier (in element 1) for guide g.